INRIA

CENTRE DE ROCQUENCOURT

Rapports de Recherche

N° 448

REAL-TIME LOCAL AREA NETWORKS: SOME DESIGN AND MODELING ISSUES

Gérard LE LANN
John F. MEYER
Ali MOVAGHAR
Simone SEDILLOT

Octobre 1985

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
BP 105
78153 Le Chesnay Cedex
France

Tél. : (1) 39 63 55 11

REAL-TIME LOCAL AREA NETWORKS :

SOME DESIGN AND MODELING ISSUES

Gérard Le Lann*
John F. Meyer**
Ali Movaghar**
Simone Sedillot*

September 1985

* INRIA, Project SCORE
** University of Michigan, Department of Electrical Engineering and Computer Science, USA

ABSTRACT

Real-time computing systems have been with us for many years. Recently, more attention has been paid to **distributed** real-time computing systems where the function of **message-passing** plays a key role. We analyze message-passing in local area networks (LANs) that are intended for such distributed real-time computing systems. Of course, real-time LANs must meet requirements identified for real-time systems in general. These requirements are presented. Current approaches to LAN architectures and protocols are examined and their limitations with respect to real-time environments are discussed.

The next part of the report is devoted to an informal description of LYNX, an experimental real-time LAN which utilizes redundant passive media. This is followed by a description of research performed to date on modeling techniques that can accommodate various features of LYNX and ultimately provide a means for evaluating its ability to perform (performability) in specified application environments.

RESUME

Les systèmes informatiques temps-réel automatisés nous sont familiers depuis des années. Depuis peu, l'attention s'est précisément portée sur les systèmes informatiques temps-réel distribués dans lesquels la fonction de communication par message joue un rôle fondamental. Nous analysons la communication par message dans un réseau local destiné à être l'un des composants d'un système réparti temps-réel. Bien évidemment, les réseaux locaux temps-réel doivent respecter les contraintes connues des systèmes temps-réels en général. Ces contraintes sont présentées. Les approches actuelles en matière d'architectures et de protocoles pour les réseaux locaux sont examinées et leurs limitations vis-à-vis d'un environnement temps-réel sont discutées.

La deuxième partie du rapport est consacrée à une description de LYNX, un réseau local temps-réel expérimental utilisant un support physique redondant. Enfin le rapport présente une description de l'état de nos recherches, à ce jour, en ce qui concerne les techniques de modélisation permettant d'une part, de prendre en compte différentes caractéristiques de LYNX et, d'autre part, de fournir les outils nécessaires à l'évaluation de sa performabilité dans des environnements donnés.

Part
John
Since
parti
schec
of th
Mova

We a
provi
Chris
schec
provi
progr
Final
use o

1. INTRODUCTION

1.1. Development of LANs and multiple access protocols

In 1964, when Rand Corporation delivered to the U.S. Department of Defense their final report entitled "On Distributed Communications" | BAR64 |, only a few people had an idea of how digital communication was to impact our world by the end of this century. A few years after the DOD-Arpanet was started (1969) and became a tool used daily by thousands of people (1974), public data networks were installed in a number of countries.

These data networks are built out of existing analog telephone networks and are intended to offer reliable communication services across media over large distances. More recently, the need for offering identical services over short distances has been widely recognized. The major driving force in this area has been Xerox which developed a local area network, called the Ethernet | MET76 |. In 1976, several versions of it were installed and used in a number of Xerox locations for the purpose of experimenting with purely digital communication technology and assessing its usefulness in the perspective of office automation.

Since then, and because of the importance of the office automation market, local area networks (LANs) have mushroomed. The two major LAN manufacturers in 1982 have been Sytek (Localnet) and Ungermann-Bass (Net-One). Most LANs developed to-day are aimed at either office automation or time-shared applications.

Recently, a few manufacturers have announced LANs that are intended for industrial applications where real-time constraints are to be met. Although presented as being specifically designed for demanding automated environments, most of these LANs are in fact either very similar to "conventional" LANs or based on well-known but restrictive techniques, such as static time-division multiplexing.

Interestingly enough, it turns out that manufacturers of such LANs are also their own customers in the sense that these LANs were developed as in-house products, needed to automate applications being taken care of by subsidiaries of such large companies as Gould-Modicon, Allen-Bradley Schlumberger, CGE, General Electric, Westinghouse, etc.

In 1980, the IEEE set up a Technical Committee, known as the 802 Committee, whose purpose is to establish standards for local area networks. The standards currently being considered by this committee are general enough so as to accomodate all kinds of technologies and topologies (passive/active, metallic/optical media) and two major classes of multiple access protocols : carrier-sense multiple access (CSMA) and token-passing (TP) protocols.

(i) CSMA-CD protocols

The principle of CSMA protocols, such as those used in the Xerox Ethernet, is to allow stations to compete freely, in a decentralized way, for gaining control of the shared communication channel when they have a message waiting for transmission. Before transmitting, a station senses the channel. If it is sensed busy, local message transmission is delayed until the on-going communication completes. If the channel is sensed idle, local message transmission is started. Several stations may sense the channel idle simultaneously. Their messages will then "collide". CSMA-CD (collision detection) protocols imply that stations listen to the channel while they are transmitting, allowing them to detect possible collisions.

When a collision is detected by a station, this station jams the channel for some time, making sure that every other station has noticed the occurence of a collision, and then selects a random number on some interval, which represents the time this station has to wait before resuming transmission. It is expected that the probability for repeated collisions to occur decreases rapidly with time. Clearly, one cannot assert that such a probability will ever be zero as new messages keep on arriving all the time. This inherently probabilistic behaviour of CSMA-CD protocols is considered as being a serious problem by many real-time computing systems designers and customers.

(ii) Token-passing protocols

The principle of token-passing protocols (on busses, on rings) is to avoid competition among stations, i.e., to avoid message collisions. A special mark (e.g., specific format of a message, special electrical signal), which is unique, is continually circulated among the stations. Stations are assumed to be arranged in such a (logical) way that each has a well identified predecessor and a well identified successor. Very simply, it is the station which possesses the token that is allowed to transmit a message on the channel. After doing so, this station passes the token to its successor. Early proposals for these kinds of multiple access protcocols can be found in | FAR73 | for physical rings and in | LEL77 | for virtual rings. In 1982, IBM submitted documents to Committee 802 describing a ring based token-passing LAN. When comparing TP and CSMA-CD protocols, token-passing protocols are usually regarded as better suited to real-time LANs because of their inherently deterministic behavior.

In our view, however, further research and experience is required to either substantiate or refute this claim. Indeed, the work described in this report is based on the conjecture that CSMA-CD protocols, with appropriate modifications for real-time applications, can provide a better solution than either conventional CSMA-CD or token passing for the networks under consideration (see Section 2.2).

Let us now introduce the IEEE 802 model as it stands presently. We will refer to this model in the remaining of the report.

1.2. The IEEE 802 reference model

The IEEE 802 Local Area Network Reference Model (LAN-RM) provides standards for the interconnection of stations [IEE82]. Each of the interconnected stations is modeled by a layered architecture.

The IEEE 802 Committee objective is to determine, for each layer, specific functions and specific interfaces with the adjacent upper and lower layers in terms of network communication. The LAN-RM layered architecture is theoretically compatible with the ISO Reference Model for open system architectures [ISO80] although the latter is wide-area-network oriented. However, three layers are used in the LAN-RM to represent the two lowest ISO-OSI-RM layers.

Each layer of the LAN-RM offers services to the upper layer and uses services from the lower layer. Services are obtainable through access points and via such primitives as transmit, receive, acknowledge, etc. In each station, the specific functions of a layer are performed by the layer entity. Protocols are defined that organize the dialog between peer-to-peer entities, that is, two entities of the same layer in two different stations.

The LAN-RM specifies three layers : the physical layer, the medium access control layer and the logical link control layer (Figure 1.1).

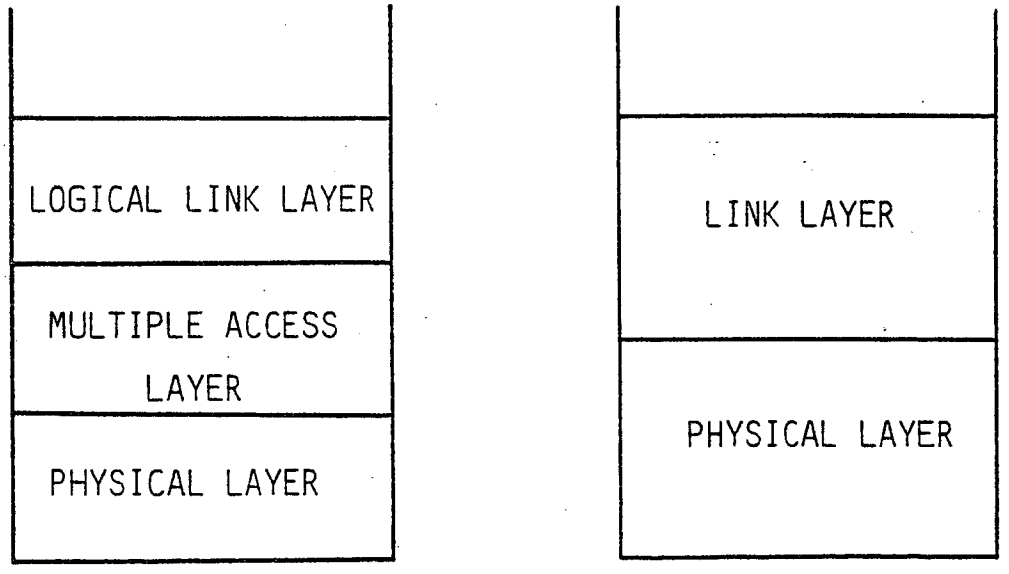

Figure 1.1 : The LAN-RM and the ISO-OSI-RM layers

(a) The physical control layer provides the capability of transmitting and receiving bits between two medium access control entities. It has the responsibility for signal modulation and demodulation.

(b) The medium access control layer (MAC) provides access to the physical layer. The LAN-RM supports both CSMA-CD and token-passing protocols. CSMA-CD protocols are derived from the Ethernet protocol while token-ring protocols are closely related to the IBM token ring proposal. Moreover, the LAN-RM supports token-bus protocols.

(c) The Logical Link Control layer (LLC) provides two types of link services : connectionless (type 1) and connection oriented (type 2). In type 1 operation, frames are exchanged between LLC entities without the need for the establishment of a logical link between two LLC Service Access Points. In connectionless operations, the LLC entity does not acknowledge the received frames. The communicating entities have no flow-control offered. In type 2 operation, a logical link is established between pairs of LLC Service Access Points prior to any exchange of user data. In the data transfer mode, frames are transmitted/delivered in sequence. Error recovery is provided. The peer-to-peer LLC protocol is derived from the HDLC asynchronous unbalanced mode as defined in the ISO 62.1979 procedure.

2. REAL-TIME COMMUNICATIONS

2.1. General requirements

Due to constraints imposed by real-time applications, there is a need to distinguish between transactional computing systems and real-time computing systems. Consequently, we will emphasize the differences in the requirements for LANs intended for these two categories of computing systems.

The first group of requirements relates to a system's ability to perform during utilization, i.e. its "performability" | MEY78 |, | MEY80 |. The second group relates to a system's adaptability to changes in application requirements and technology.

2.1.1. Performability

The performability of a system involves several aspects of system behavior, including the following.

(i) Correctness

Assume that some metric is defined, that allows for the evaluation of how accurate the content of an output state is. It is better for a transactional system to produce an inaccurate state (the extreme case being the triggering of an exception or a catastrophe) rather than not producing a state at all, for it is expected that subsequent detection and "compensation" can take place. Conversely, a real-time system must produce accurate output states only. Because external physical phenomena have some intertia and because it is impossible to transparently recover from the effects of an incorrect output, it is better for a real-time system to be exceptionnally silent in case of error rather than to produce inaccurate outputs, exceptions of catastrophes. Since the correctness of an output depends on its timing, it is also necessary to characterize the notion of promptness.

(ii) Promptness

Assume the existence of a physical time reference that is common to a system and to its physical environment. It is better for a transactional system to produce an output even if some expected response time is exceeded. The environment is assumed to be able to tolerate (possibly largely) varying and theoretically unbounded response times. Conversely, it is useless (and dangerous) for a real-time system to deliver an output if some deadline has been missed. Real-time systems must be able to either deliver correct outputs within strict time margins or remain exceptionnally silent.

(iii) Reliability and availability

Both of these terms have received widely accepted definitions. Reliability and availability can be enhanced by the use of fault prevention and fault tolerance techniques. The use of fault tolerance techniques is based on the generally accepted belief that a complex system, no matter how carefully designed and validated, is likely to contain residual design faults and to encounter unpreventable faults during its operation. Utilization of real-time systems in hostile environments and impossibility to maintain them/repair them at will are the prime motivations for stringent requirements in this area.

In real-time applications, erroneous computation (in the algorithmic sense) and erroneous timing (in the chronological sense) are the enemy. A valid computation, if performed too late, might result in faulty behavior of the system. The consequence of this observation is that reliability/availability issues and timing issues cannot be addressed separately. Assumptions regarding the nature and the frequency of faults that should be tolerated, as well as accepted probabilities of faulty behaviour need to be stated rigorously at each level of abstraction.

2.1.2. Flexibility

In the course of a recent study [SCI83], we have identified four types of flexibility requirements.

(i) Functional flexibility

For any level of abstraction, functional flexibility means that it is possible to completely identify which modules will be affected by a change in a functional requirement and to obtain the new specifications (interfaces and protocols) of these modules. If it takes too long to conduct these tasks, then flexibility is not practically achieved. Therefore, it would be very useful to automate these tasks. This is still an ideal goal today.

(ii) Implementational flexibility

This kind of flexibility is needed when it is desired to adapt the way a system is implemented to technological changes for economic reasons, for performance considerations, for practical reasons or because of changes in the functional specifications. Replacement and/or addition of modules over time raise the issue of heterogeneity in hardware, software and data representation. This stresses the importance of well defined interfaces and protocols.

(iii) Geographical flexibility

The design of a system should be such that there is no a priori artificial limitation regarding geographical dispersion of the hardware, the software or data. The only limitations to be taken into consideration are those derived from Physics (e.g., speed of light). Clearly, geographical flexibility imposes constraints on how functions are implemented. For example, choosing a carrier-sense multiple access protocol to operate a passive medium yields upper bounds on the distance between any two communicating modules. Geographical flexibility implies that discrete or continuous changes in the system configuration should not require an "instantaneous" replacement of a given set of modules.

Requirements presented so far could be characterized as "static" flexibility requirements in the sense that the three logical steps "system observation", "decision", "materialization", are conducted from outside the system and can be separated in time by arbitrarily large delays. There is a need for identifying another type of flexibility requirements, namely operational flexibility.

(iv) Operational flexibility

Operational flexibility refers to a notion of "dynamic" flexibility requirements which state that, at any time, a system should be able to adapt itself to changing conditions (e.g., traffic peak loads, faults) that is, the three logical steps mentioned above are conducted from within the system so as to maintain some performability level. Algorithms that dynamically achieve resource allocation, process scheduling, error detection/recovery or masking meet operational flexibility requirements.

Clearly, performability and flexibility requirements have strong relationships in the case of real-time systems. This lack of "orthogonality" is one of the most serious problems to be tackled when specifying and designing such systems. Some formalism is needed for stating the time-independent properties (called safety properties) and time-dependent properties (called liveness properties) that are sought. Of paramount importance are liveness properties which include real-time constraints. We want to identify and analyze LANs and protocols which meet all of the above requirements.

2.2. Selection of a real-time LAN

It is unreasonable to envision a single, general purpose LAN that would be suited to all kinds of traffic patterns. There is no "best" computer that is ideal for all possible applications. Similarly, there is no "best" LAN.

In the context of our research, we have been led to analyze and compare the various technologies, topologies and protocols that could be envisioned before defining the general architecture of a real-time LAN which would meet our requirements. The considerations which have led us to define such a LAN, referred to as LYNX (Local Industrial Network Xperiment), are presented below.

2.2.1. LYNX topology and technology

Our approach is very much in agreement with the results reported in [POW81a]. Among the few topologies which are known to be best suited to support correct, prompt, reliable/available and flexible LANs, we have selected a redundant broadcast passive bus topology. One of our motivations is that it is possible to interconnect a large number of stations (several hundreds) on such a topology without running into the problems encountered with such topologies as rings for example, whether physical media used are metallic coaxial cables or optical fibers.

From now on, physical media will be referred to as **channels.** If cost effective and not in contradiction with reliability/availability requirements, channels can be implemented as transmissions at different frequencies on a unique physical medium, provided that stations are equiped with a n-frequency hardware interface. Otherwise, channels are implemented as separate physical media, each of which operating at some unique frequency.The technology employed should accomodate bandwidth in the order of 100/200 Mbits/s and distances encompassed should be in the order of several hundreds meters, up to 1 km roughly.

2.2.2. LYNX multiple access protocols

Depending on the type of activities conducted by the processes which utilize stations to communicate with each other, one is led to establish a set of different traffic assumptions. As a result, it is possible to select appropriate protocols in each case.

Messages to be transmitted over LYNX may be short (e.g. control and monitoring messages) or long (e.g. computer files). The sizes considered range between a few bytes to several millions of bits. Messages may also be submitted periodically or aperiodically. For handling periodic traffic whose period is very short, a special-purpose LAN operated under a static time-division multiplexing protocol might be appropriate. Such protocols have been analyzed extensively and will not be considered further. LYNX is intended to handle efficiently aperiodic traffic (short, medium and long messages) as well as periodic traffic (any size message) when message periods are not too small.

As mentioned in the introduction, two types of multiple access protocols are currently being considered by the IEEE 802 Committee : token-passing (TP) protocols and carrier-sense multiple access (CSMA) protocols. The conjecture underlying our research work is that CSMA-like protocols are best suited to handle real-time traffic, provided, among other things, that a solution is found whereby message collisions are resolved in a deterministic manner.

We will not examine here the respective merits of TP versus CSMA protocols with respect to reliability/availability requirements or flexibility requirements. Although these are important requirements, the central issue is how the two types of protocols compare with respect to promptness (timeliness) requirements imposed by real-time constraints.

2.2.2.1. Real-time considerations

To begin with, let us explain why some designers believe that real-time communication problems are a "myth". The reasoning goes as follows : measurements performed on existing (office oriented) LANs demonstrate that the average load on such LANs is limited, in the order of 5 % - 10 % of the available bandwidth. Therefore, there is no need to devise sophisticated multiple access protocols because all messages should be easily transmitted in time under such assumptions, using conventional TP or CSMA-CD protocols. This reasoning is correct, given the assumptions hold. Unfortunately, the assumptions are totally erroneous. Specifically, we do not believe that :

(i) measurements conducted in office environments are illustrative of what could be measured in an automated factory (as an example)

(ii) observations made during the early days of office/manufacturing automation cannot be assumed to remain unchanged when the degree of automation increases. Who predicted traffic jams on hudge highways at the time Ford introduced the Model T car ?

Moreover, reasoning in terms of average values, measured on time intervals which are orders of magnitude bigger than the sampling period of some external phenomenon to be controlled, is totally misleading and has no relevance at all in a real-time context. It is indeed the case that traffic peaks exist on time slots as small as 10 or 100 milliseconds. It is indeed the case that such traffic peaks should be absorbed in time by a real-time LAN, even if the instantaneous utilization ratio is high (e.g. 70 %). It is indeed the case that real-time LANs must be able to absorb avalanches of "emergency" messages in time. Clearly, it is precisely under such "difficult" circumstances that the behavior of a real-time LAN must be predictable and proven to meet promptness requirements.

2.2.2.2. The need for further study of protocols

Let us now look at the TP versus CSMA protocol controversy. The most obvious advantage of TP protocols with respect to promptness requirements is their "deterministic" behaviour, in contrast with CSMA-CD protocols which are inherently probabilistic. By looking more carefully at TP protocols, one is led to conclude that the "deterministic" behaviour of TP protocols depends very much on specific assumptions. For example, it is possible to compute an upper bound for channel access times with TP protocols provided that :

(i) the maximum number of stations is known, the maximum time spent by the token in a station is known, the maximum transmission time of the token between two stations is known,

(ii) single and multiple faults/failures do not occur.

Of course, assumptions (i) and (ii) are in contradiction with flexibility and reliability/availability requirements. Rejection of these assumptions yields the conclusion that the LANs we are interested in are probabilistically deterministic, even when operated under a TP protocol. In particular, maximum channel access times, as achieved by TP protocols, can never be guaranteed with absolute certainty. Another unpleasant feature of TP protocols is the existence, under light load conditions, of a minimum average channel access time that is non negligible compared with ones achieved by CSMA protocols. This is a result of the assumption which underlies the concept of TP protocols, i.e., that at any instant of time, there is a non-zero probability that at least two stations want to have access to the channel. When this is not the case, TP protocols perform poorly compared with CSMA protocols.

Another serious practical problem with TP protocols occurs when the upper bound on access delay exceeds the maximum access time for messages requiring rapid transmission, e.g. urgent commands. In this case one must either reject the use of TP or use it with the understanding that system behavior is no longer deterministic, i.e. there is a non zero probability of experiencing excessive access delays.

Of course, the above observations are not meant to suggest that TP protocols are always bad and that CSMA protocols are always good. What we believe is that, in many circumstances, the constraints inherent to TP protocols are not compensated by a sufficiently attractive behavior, in comparison with CSMA protocols. Obviously, the probabilistic behavior of CSMA-CD protocols becomes a serious problem under medium and heavy load conditions as regards promptness requirements. Consequently, desired multiple access protocols should retain the advantages of CSMA-CD protocols under light load conditions and the advantages of TP-like protocols in other conditions, i.e. when the probability of message collisions is not negligible. In other words, what we seek is a deterministic collision resolution scheme (CSMA-DCR). The work reported here is a contribution to the vast effort undertaken by different teams in the area of CSMA-DCR protocols [CAP79], [CHL79], [CHL80], [FAY83], [FRA80], [GOL83], [KUR83], [LEL83b], [MAS81], [MOL81], [POW81b], [ROM81], [TOB80], [TOB82], [TOW82], [VAL83].

Like most of these studies, the CSMA-DCR protocols presented are based on two building blocks which are hardware implemented :

- the end-of-carrier (EOC) signal
- the end-of-jamming (EOJ) signal.

One of these protocols makes use of the channel slot time, whose value will be referred to as a. The only timing computations using slot time a that we allow ourselves are based on conventional features, as those used in the Ethernet-compatible Intel 82586 chip.

2.2.3. LYNX message scheduling policy

Intuition and queueing theory indicate that promptness requirements cannot be met in general when scheduling algorithms or servicing policies are based on static priorities. In a real-time computing system, one finds many reasons to make explicit use of physical time. Consequently, physical clocks are to be taken into account in the early design stages. This is our approach for LYNX. Message scheduling will be conducted according to time oriented priorities. More precisely, every message will be carrying a physical timestamp indicating its transmission deadline. These deadlines will be used to perform deadline dependent scheduling and time dependent scheduling. It is also advantageous to check at various points before transmission whether a message has missed its deadline and, if so, exclude it from futher processing via rejection. This feature is referred to as **promptness control.** The servicing policies that are part of LYNX multiple access protocols will make explicit use of deadlines to enforce promptness properties, which indicates that they will almost never be equivalent to LIFO or FIFO strategies. Furthermore, they must be able to naturally accomodate emergency messages, i.e. messages that must be transmitted between two or more stations at maximum speed, regardless of on-going activities. These policies, to be efficient, require hardware implementation of such primitives as those selecting a customer satisfying a criterion in a waiting queue and such tools as timers. Recent hardware products, e.g. the Intel iAPX 432 microprocessor, suggest that such implementations are well within the possibilities of current technology.

3. INFORMAL DESCRIPTION OF LYNX

3.1. General description of LYNX

The LYNX model that is given in Figure 3.1 is intended to be used as a vehicle for an examination of the problems to be tackled and research to be undertaken (see Section 3.4). The model is consistent with the IEEE 802 model presented in section 1.2. and was initially described in | LEL83a |.

A station comprises two entities : a source entity (S) and a destination entity (D). Messages deposited in the Unscheduled Queue of a source (UQ(S)) are processed first come first served (FCFS). The source link level process (SLP) establishes some ordering on the set of messages, reflected in the Scheduled Queue SQ(S). Similarly, the destination process (DLP) transforms UQ(D) into SQ(D). As we assume passive communication channels, messages received by the destination multiple access process (DMAP) are served FCFS.

Each station (source/destination) is provided with a physical clock which is used to assign timestamps and deadlines to messages. We assume that arbitrarily good clock synchronization is achieved via some message exchange mechanism that meets the requirements presented under Section 2.1. Such mechanisms have been proposed and verified | LAM82 |.

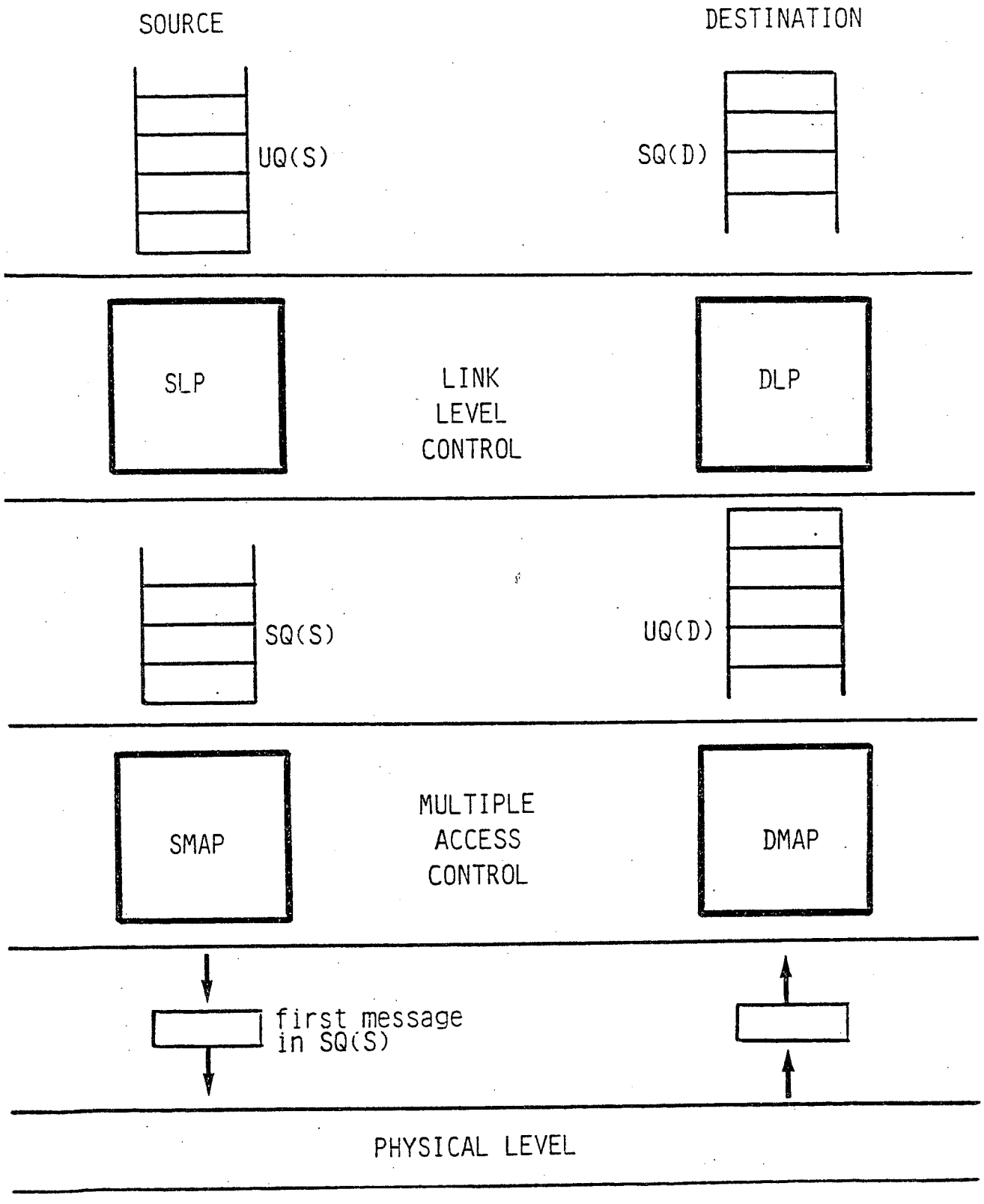

Figure 3.1 : Model of LYNX
(only one source and one destination are represented)

The two levels we focus on correspond to level LLC and level MAC in the IEEE 802 Reference Model. As mentioned before, the problems we analyze are slightly different from those addressed by the 802 Committee in the sense that we want to devise algorithms and protocols which ensure that messages are scheduled and transmitted by sources so as to arrive in time at their destination. Note that this is not necessarily equivalent to ensuring fairness or equal sharing of the bandwidth among competing stations, nor to ensuring optimal bandwidth utilization.

As we are mainly interested in message scheduling and message transmission, we will concentrate on only one primitive, namely the **SendMessage** primitive.

For an arriving message m, the goal is to ensure (with high probability) that the execution of **SendMessage** is completed within a specified time duration E(m).

E(m) is either a constant, part of the Link Level specifications, or a parameter provided by every process invoking the Send Message primitive at this level. In the latter case, the parameter value must be consistent with the expected system performance. In both cases, E(m) may depend on a number of attributes, such as message length, message type, etc.

E(m) can be expressed in terms of the following time durations :

- a(m), sojourn time in UQ(S) and in SLP,
- b(m), sojourn time in SQ(S) and in SMAP (influenced by collision resolution),
- c(m), sojourn time in the channels and in DMAP,
- d(m), sojourn time in UQ(D) and DLP.

Sojourn time in SQ(D) depends on the scheduling strategy used at a higher-level. It is not included in E(m) since link level processes have no control over this sojourn time. The protocols and algorithms we consider can influence b(m) and d(m).

Let us now examine the following issues :

- message scheduling, local and global
- promptness control
- channel multiple access control.

3.2. Message scheduling

As indicated under Section 2.2.3, we are led to consider deadline oriented and time-dependent priorities. Fixed priorities have the serious drawback that only highest priority messages can be guaranteed some particular service (e.g. prompt transmission). Before presenting the dynamic priorities that are used at the Link Level and describing how such priorities are used at the MAC level, let us elaborate on the concept of emergency messages.

It is obvious that in LYNX, there is no need to define various message categories in order to attach priorities to messages. We are then dealing essentially with only one category of messages. However, under specific circumstances whose occurences are **exceptional**, we want to allow stations to assign some messages transmission deadlines that are very close or equal to current time and we want to allow stations to exclude the use of promptness control for such messages.

For practical reasons, these messages will be referred to as **emergency messages.** All other messages will be referred to as **regular messages.** We assume that, on the average, an emergency message submission is a very rare event. However, on a short time slot, LYNX may have to handle several emergency messages simultaneously (an avalanche). Also, we intend to employ the same scheduling strategy for both emergency and regular messages. Consequently, the perturbations created by exceptional emergency messages are viewed as being negligible, except in the case of an avalanche.

This approach, which lends itself better to analysis (see Section 4), is to be contrasted with an approach where one would be dealing with two categories of messages, each of which being associated different static priorities. When an avalanche (exceptionally) occurs, it is the objective of LYNX to process it first, possibly to the detriment of some regular messages.

3.2.1. Link level, local scheduling

SLP could simply timestamp every message m arriving in UQ(S) with the time read on the local clock, denoted t(m), and then transfer m into SQ(S). In this case SQ(S) = UQ(S). Timestamps are used by DLP to build SQ(D) according to increasing values of timestamps contained in messages received from many sources. If we now assume that we can establish an upper bound d for d(m), and that we know how to compute c(m), SLP can compute T(m), the deadline for transmitting message m successfully on a channel, as follows :

$$T(m) = t(m) + E(m) - c(m) - d.$$

Messages in SQ(S) are ordered by increasing T(m) values. New messages may be inserted anywhere in SQ(S) by SLP at any time. It is often the case in real-time systems that operations are assigned fixed priorities. Consequently, a real-time LAN must be able to handle such priorities as well. A way of combining fixed and deadline oriented priorities consists in defining a choice function H such that :

$$H(m,t) = j(m) + h(m) * dt(m)$$

with

j(m) = static priority for message m, a non negative integer with 0 as the highest priority,

dt(m) = T(m) - t, where t is interpreted as current time,

h(m) = positive constant.

Messages in SQ(S) are ordered by increasing H(m,t) values i.e., if messages m1 and m2 are such that H(m1) < H(m2) then m1 is served before m2. This is the kind of priority used, for instance, in the computing systems which operate French nuclear submarines | DER67 |.

Let us first consider the case where h(m) is the same for all m, i.e., h(m) = h. Then H(m) induces an ordering on every set of "simultaneous" messages that is time independent. Consider two messages m1 and m2 such that H(m1) is computed by SLP before H(m2). One can have either m1 precedes m2 in UQ(S) or m2 precedes m1, which is the case when the following inequation holds :

$$j(m2) < j(m1) - h * | T(m2) - T(m1) |.$$

As we see, current time t does not intervene in this inequation.

In general, however, when values of h(m) differ for different messages, it is possible for H(m,t) to combine both fixed and time dependent priorities.

More specifically, for two messages m1 and m2 such that j(m1) < j(m2) and T(m2) < T(m1), one must find h(m1) and h(m2) such that there exists an instant Z satisfying

$$\max | t(m1), t(m2) | < Z < T(m2)$$ and yielding the following property :

For all values of t satisfying $\max | t(m1), t(m2) | < t < Z$, we have

$$H(m1, t) < H(m2, t)$$

and, for all values of t satisfying Z < t, we have

$$H(m2, t) < H(m1, t)$$

See figure 3.2 for a graphic representation. More on priorities can be found in Section 4.

From now on, we will assume deadline oriented priorities at the link level, with the first message in SQ(S) being assigned the lowest deadline value.

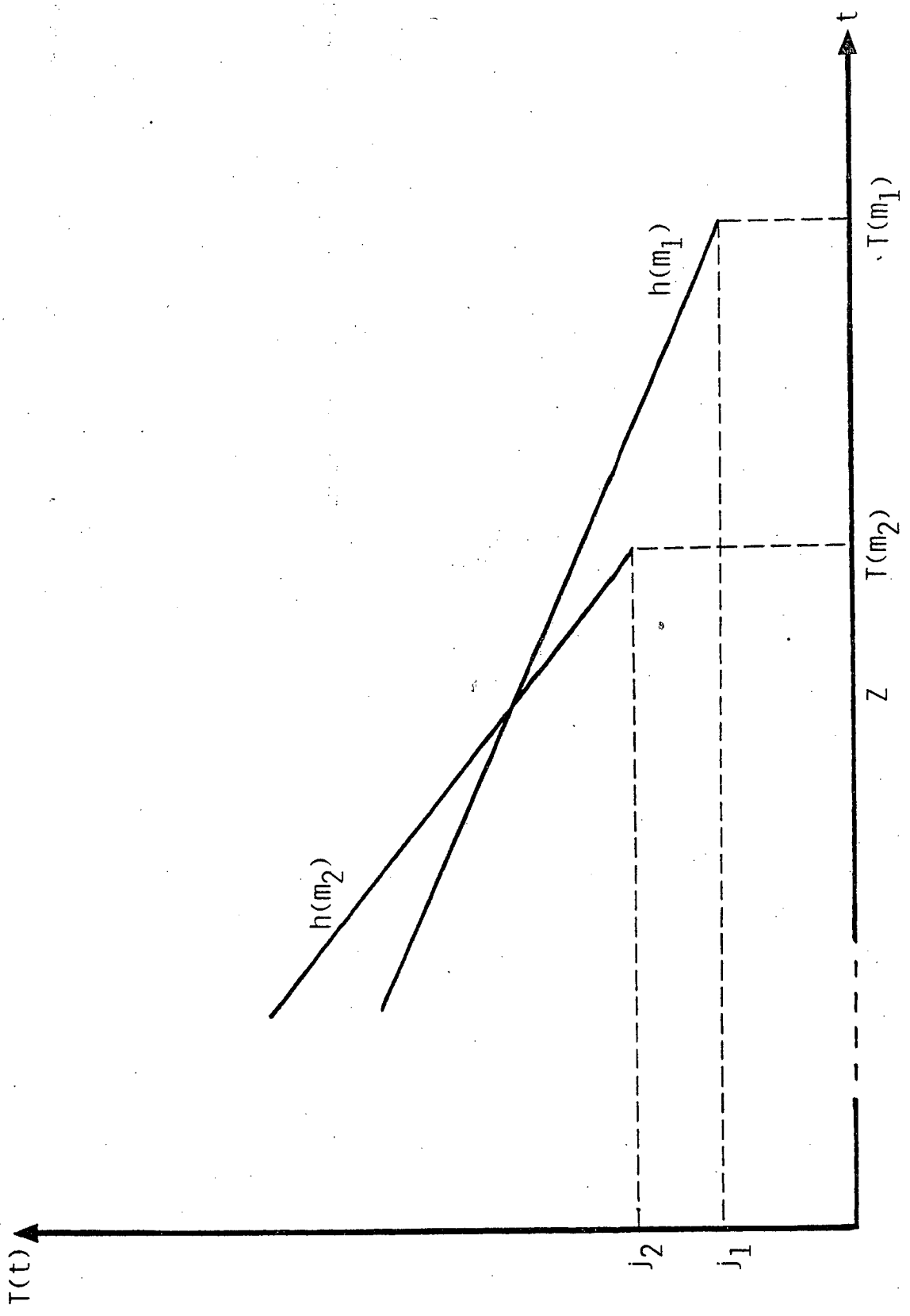

Figure 3.2 : Time-dependent priorities

3.2.2. MAC level, global scheduling

At the MAC level, we are concerned with global message scheduling. We want to minimize rejection of messages due to promptness control (see below). Consequently, we will try to resolve competition among message sources in such a way that messages are transmitted in the order defined by increasing deadlines, system-wide. The more global this information, the better this objective will be achieved. However, the requirements introduced in Section 2.1 force us to use as few global variables as possible. In our current research, the objective is to restrict ourselves to use only one global variable, namely current physical time.

When a message arrives at the head of SQ(S), in terms of current time t, we use an algorithm to determine at which instant T ($t \leqslant T \leqslant T(m)$) the message transmission will be attempted. The computation is as follows :

$$T = t + K * | T(m) - t |$$

where the value of K is in the range $0 \leqslant K \leqslant 1$. Parameter K could be assigned a unique constant value or could be assigned different values at different sources in order to adapt to changes in local traffic conditions. The role of parameter K is examined in Section 4.

Conditions for computing T

Consider source S which has messages waiting in SQ(S). Various conditions can be envisioned under which this computation is performed.

They are related to the behavior of the SMAP which acts as the server of SQ(S). A general requirement is that when time T has been reached, SMAP is not preemptible (i.e. no other message will be serviced until the message currently serviced by SMAP has been transmitted or rejected). For the sake of simplicity, we present two possible servicing strategies only.

i) Non-preemptive service

SMAP can enter two states, free and busy. The condition for activating service for message m is as follows :

(m first in SQ(S)) AND (SMAP free).

Let t be the time at which this condition becomes true ; it is at time t that T is computed for m. More precisely, at time t :

- SMAP enters state busy
- m is taken out of SQ(S) so to be served by SMAP
- T is computed.

No preemption of SMAP is authorized between instant t and instant T. At time T(m) + c(m) at the latest, SMAP returns to state free.

ii) Preemptive service

SMAP can enter the following states :

- free,
- waiting (m') : a message m' (whose deadline is T(m')) is being served by SMAP and has been taken out of SQ(S). SMAP remains in this state until time T', the time at which the first attempt for transmitting m' is performed,
- busy (m'), when time T' is reached.

SMAP returns to state free at time T(m') + c(m') at the latest.

The condition under which service for some message m will be granted is as follows :

((m first in SQ(S)) AND (SMAP free)) OR
((m first in SQ(S)) AND (SMAP waiting (m')) AND (T(m) < T(m'))).

Let t be the instant at which this condition becomes true. At time t :

- m' is put back into SQ(S), if needed
- m is taken out of SQ(S)
- SMAP enters state waiting (m)
- T is computed for message m.

The research on modeling issues reported in section 4 assumes non-preemptive service.

3.2.3. Promptness control

Obsolete messages should not be transmitted. In addition to wasting resources, transmission of obsolete messages could be dangerous for the application. Messages can become obsolete either because they arrived "late" in UQ(S) or because a(m) is too large. In fact, promptness control is performed at all levels, at the source and at the destination. Promptness control ensures that, in the computation of T (see above), $t \leq T(m)$. Promptness control and its influence on the management of SQ(S) are analyzed in Section 4.

3.3. Multiple access protocols

Two multiple access protocols are presented in | LEL 83b |. They are believed to meet the requirements presented under Section 2.1 (although it is not demonstrated that they do so). They are designed so as to operate on a multiplicity of channels, in a transparent manner, i.e. in such a way that faults and breakdowns of some of the LYNX elements are kept invisible to higher levels as long as a minimum amount of such elements remain active and fault-free.

As indicated in that report, research on deterministic multiaccess protocols is being pursued. Consequently, one should not assume that implementation of LYNX will be restricted to one of these two protocols.

3.4. Research goals

Through investigation of LYNX, the goals of this research are to :

1) Acquire a better understanding of how real-time constraints can be accomodated in the design of LANs (IEEE 802 levels LLC, MAC).

2) Identify performance, reliability and, generally, performability measures that are appropriate for evaluating real-time LANs.

3) With respect to measures identified in 2), develop analytic and simulation models that can account for complex features of the proposed system ; specifically deadline-dependent priorities, proptness control, and time-dependent scheduling.

4) Using the modeling Techniques developed in 3) , evaluate LYNX and, particularly, the effects of proposed features (see goal 3) on the system's ability to perform.

The research reported in the section that follows is the initial phase of the activity required to accomplish these goals. Previous research that led to the definition of LYNX (including the algorithms/protocols it incorporates) has already provided a better understanding (in intuitive terms) of how real-time constraints can be accommodated (Goal 1). However, much work, of a more substantive and quantitative nature, remains to be done in this regard. Indeed, the desire to substantiate such knowledge in more scientific terms is a principal motive for pursuing goals 2) - 4).

Regarding the identification of appropriate measures (goal 2), we contend that usual measures of LAN performance (average throughput, average delay, channel utilization, etc.) do not generally convey the type of information that is relevant to the use of real-time LANs. Some of these measures are simply not appropriate (e.g., average delay) ; others require modification (e.g., average throughput). This is due primarily to promptness requirements (see Section 2.1) which alter the view of what constitutes successful transmission of a message. In the real-time case, a message is transmitted successfully if it is received and reception occurs within specified time limits. Accordingly, the measure "average delay" (between beginning of transmission end of reception) conveys little (it is debatable as to just how little) about the performance of a real-time LAN. A much more significant measure is the proportion (under steady state conditions) of attempted message transmissions that are successfully received, the latter implying that promptness requirements are met. This measure (see Section 4.1 for details) has emerged as the principal performance measure for the system in question. More refined information concerning the ability to meet deadlines is also interesting, but is generally more difficult to obtain. For example, one might consider the probability distribution, among all messages successfully transmitted, of the difference between a message's due date and its actual time of reception. A second important measure, identified in connection with goal 2), is "effective" throughput, i.e., the average rate (in steady state) at which messages are successfully received. Note that this measure also reflects the system's ability to satisfy promptness requirements (whereas the usual notion of throughput does not).

In addition to measures of (strict) performance, we are interested, more generally, in measuring the system's ability to perform in the presence of faults, i.e. its performability. The performance measures discussed in the previous paragraph can also serve as measures (performance variables) for

performability evaluations. In addition, other measures, more directly related to fault occurrences, should be considered. However, significant differences between performance versus performability arise in the modeling process (goal 3) rather than in the definition of appropriate measures.

Model development (goal 3) appears to be the most challenging of the prescribed goals. A review of existing literature on the performance modeling of LANs indicates rapidly increasing concern with this problem, but most of the work published to date does not address real-time issues.

Concerning (strict) performance, prior work we are aware of fails to include promptness requirements. Accordingly, the promptness control feature of the proposed system is something that, to the best of our knowledge, has not been accounted for in previous modeling studies. A second system feature that complicates the modeling process is the use of deadline dependent priorities. Although these kinds of priorities have been treated in the context of operations research (see Section 4.4), the specific nature of the proposed scheduling appears to pose some new and interesting modeling problems. A third feature of the system is time-dependent scheduling at the MAC level using the algorithm described in 3.2.2. Although such scheduling has been considered in a queueing model context (see | KLE76 | for example) it is not yet clear how known results might be extended to the problem in question. The type of deference delays that result from this scheduling appear similar to delays experienced when synchronizing concurrent processes, introducing known difficulties in the representation of complex timing interactions. In short, this type of scheduling, coupled with a specific protocol for accessing the channel, comprises a process which is very difficult to model. Indeed, access protocols themselves (without time-dependent scheduling) call for relatively complex models | FRA80 |, | KLE75 |, | MOL81 |, | TOB82 |.

In addition to performance considerations, this goal is more generally concerned with obtaining an appropriate performability model of the system. However, experience (see | MEY82 | for example) has shown that performability models depend on the ability, initially, to model strict performance. Hence the work reported here has focused on modeling the (fault-free) performance of the system. (Follow-on work by Movaghar and Meyer | MOV84 | extends the example of Section 4.3.5 so as to include the possibility of faulty resources.)

The types of models we seek include both analytic models and simulation models. Work to date has focused on analytic models where it appears that queueing network models are not rich enough, structurally, to accomodate the type of properties described above. What is called for are analytic models with more detailed structural primitives (e.g., stochastic Petri nets) which are capable of representing more complex types of stochastic behavior. It is also obvious that a simulation modeling effort must be undertaken if the system is to be better understood.

Finally, evaluation of the system (goal 4) will employ the models (goal 3) to evaluate the measures identified in goal 2). Due to the type and complexity of the models we anticipate, software tools will be required to aid the evaluation process. Indeed, even in the case of relatively simple models representing promptness control (see Sections 3.2.3,4.3), we have found that solutions require the aid of special programs such as RdPS (CNAM) or recent extensions to METAPHOR (Univ. of Mich. and the Industrial Technology Institute | MEY85 |).

4. CURRENT RESEARCH ON MODELING ISSUES

4.1. Environment and measures

4.1.1. Environment

For the system in question, we assume that messages arrive at the inputs to various stations of the network according to specified distributions of interarrival times and timing requirements (deadlines or allowed delays). We believe, although this is not yet substantiated, that performability of the system (according to the measures discussed below) will be quite sensitive to assumptions about this environment, particularly those relating to differences among local workloads at each station.

Our modeling effort to date has considered mainly local behavior (in a single station) where we assume that messages arrive in a purely random fashion (i.e., as a Poisson process) and that each message has a deadline for arriving at its destination. The difference between this deadline and the time of arrival is the message's **allowed delay.** Messages are classified according to their allowed delays and we assume that the number of such message classes is finite. The probability distribution of incoming messages according to class (i.e., the probability distribution of allowed delay) is assumed to be included in the specification of the local environment. We assume further that this distribution is time-invariant, i.e., the same distribution applies any time a message arrives. No other conditions are placed on the nature of this distribution.

We have yet to consider the more general problem of specifying a global environment (workload) for the system. One possibility is to regard stations as homogeneous and independent with respect to input demands. In this case, the global environment could be defined as a collection of local environments of the type described above. Other possibilities should also be considered.

4.1.2. Measures

As informally discussed in connection with the second goal of this research (see Section 2.1.), we seek to identify performance and, more generally, performability measures that convey a real-time system's ability to perform. The following is a discussion of measures already identified.

Let $|0,t|$, where $t > 0$ denote the period during which the system is utilized and consider the (random) variables

$A(t)$ = number of messages that arrive (at the inputs to the system) during $|\ 0,t\ |$.

$RB(t)$=number of messages that are processed on time (i.e., meet their deadlines) during $|\ 0,t\ |$.

Then real-time throughput measures can be defined as follows :

$RTh(t) = \frac{RB(t)}{t}$ is the **real-time throughput** of the system during $|\ 0,t\ |$.

$NRTh(t) = \frac{RB(t)}{A(t)}$ is the **normalized real-time throughput** of the system during $|\ 0,t\ |$.

Note that NRTh(t) also expresses the fraction of arrived messages that are successfully processed (meet their deadlines) during $|\ 0,t\ |$. This variable can serve either as a (strict) performance measure or, when faults are accounted for, a performability measure. In the strict performance case, we anticipate that the system states will have a limiting (steady-state) probability distribution, whence the performance variable NRTh(t) should likewise converge to a limit. More precisely, under these conditions we define

$$NRTh = \lim_{t \to \infty} NRTh(t)$$

where by definition of NRTh(t), it follows that

NRTh = Probability (under steady-state conditions) that an arrived message is processed on time.

With respect to strict performance, we regard NRTh as an important measure of real-time system behavior. A second variable, closely related to NRTh, is the steady-state real-time throughput of the system, i.e., the measure

$$RTh = \lim_{t \to \infty} RTh(t)$$

Given that messages arrive with average rate α, i.e., $\alpha = \lim_{t \to \infty} \frac{A(t)}{t}, RTh$

is immediately determined by NRTh, i.e.,

$$RTh = \alpha . NRTh$$

More refined performance measures can also be considered. For example, let

W = system time of a message, i.e. the time between arrival at the input and departure at the output.

D = allowed delay.

(Note : So that the variable W may apply to any message, if a message is rejected by the system, then W has the value ∞.) In terms of W and D we define the variables

L = lateness = W - D

E = earliness = D - W = - L

In the case where all allowed delays are finite, it follows that $L = \infty$ (equivalently, $E = -\infty$) if and only if $W = \infty$

The probability distribution function (PDF) of L, i.e., the function

$$F_L(x) = P(L \leqslant x)$$

provides more refined information about the system performance with respect to deadlines. In particular, since a message arrives on time iff its lateness is not positive, it follows that

$$NRTh = F_L(0).$$

In the case of earliness, we will typically be interested in the conditional distributions

$$P(E \leqslant x \mid L < \infty)$$

and

$$P(E \leqslant x \mid L \leqslant 0)$$

The first describes how early messages are received, given they are indeed received ; the second describes how early messages are received, given they are received on time. Note that in the latter case (since $L \leqslant 0$ iff $E \geqslant 0$), E assumes only nonnegative values.

4.2. Local message scheduling

4.2.1. Description

As is described in 3.1. and 3.2., messages arriving at the input to LYNX are deposited in the UQ(S) queue with a first-come-first-served discipline. It is the source link level process SLP which is used to establish a local message scheduling (LLC level) ; it takes into account deadlines for transmitting the messages on the channel (T(m)), as well as fixed priorities of these messages (j(m)). The goal of this section is to study the behavior of the queue SQ(S) (priority disciplines, waiting time, etc.) with the aim of identifying problems that require further investigation.

In Figures 4.1.a and 4.1.b we describe two possible servicing strategies of the process SMAP (cf. 3.2.) which will influence the behavior of SQ(S).

First, a non-preemptive service

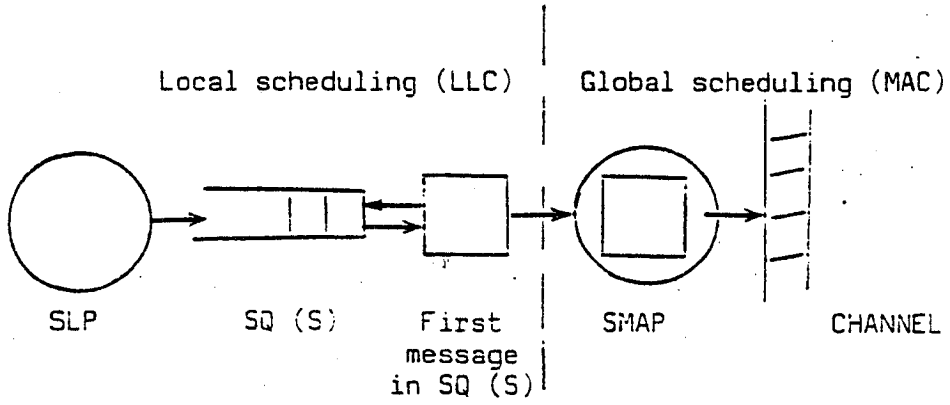

Figure 4.1.a

In this case, after each transmission of a message from the source i, the $SMAP_i$ process selects the first message of $SQ(S)_i$. This message is the only message selected by source i for channel access, without accounting for new messages which may arrive in $SQ(S)_i$ during its waiting time.

An alternative scheduling scheme is the following :

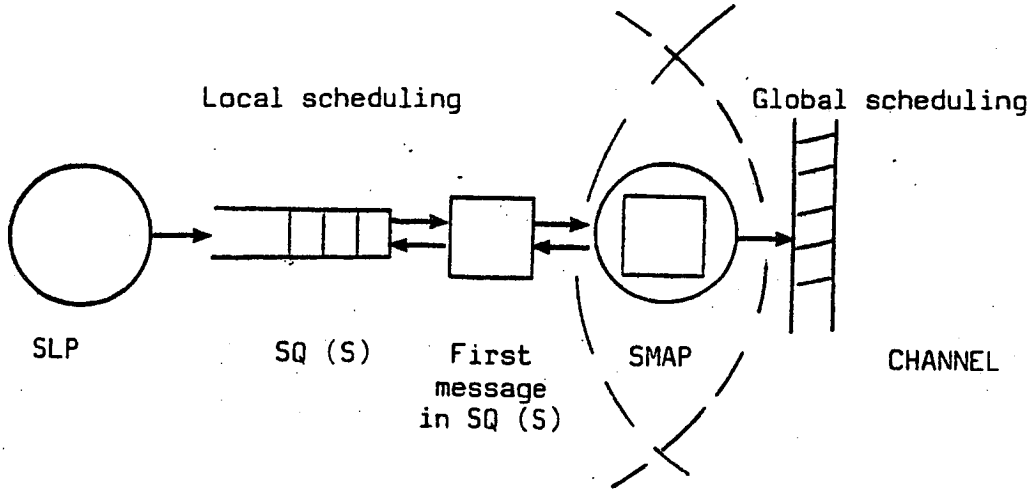

Figure 4.1.b

After a message transmission, $SMAP_i$ selects, as before, the highest priority message in $SQ(S)_i$; this message may have to wait before its own transmission on the channel (channel busy). But, during this waiting time before transmission, if a new message arrives with a higher priority, it preempts the first one which, in turn, goes back to the second place of the queue.

In the first case (Figure 4.1.a) we have a system where a priority discipline may be defined for SQ(S) and where the service time may be considered as the waiting time plus the total transmission delay on the channel. We shall assume later that this service time is exponentially distributed. Thus, we have a nonpreemptive queueing discipline (cf. next paragraph). In the second case, if we consider the service time as before, we have a preemptive discipline (service may be interrupted during its first part). On the other hand, if we regard service as transmission only, the discipline becomes a non preemptive one, but the waiting time in SQ(S) id much more difficult to appreciate. The second case (Figure 4.1.b) seems more interesting concerning the efficiency of the communication system (for example, the percentage of rejected late messages might be lower than in the first schema). In any case, there is a need to study, analytically or by simulation, the two possibilities.

4.2.2. Queueing disciplines

There is a variety of means (see | KLE76 |, for example) for choosing which message in the queue is to be serviced next. The decision may be based on

- some measure related to the relative arrival times (ex : **FIFO** (FCFS), **LIFO** (LCFS) (first or last-come-first-served)) ;

- some measure of the service time required, or the service so far received : **Service-time-dependent priorities** (ex : **SFF** (shortest-job-first)).

- some function of group membership (ex : **HOL** (head of the line priorities = fixed priority queueing). An arriving message belongs to one of a set of J different priority classes (j=1,..,J) and joins the queue behind all messages from group j (and higher) and in front of all messages from group j+1 (and lower)).

- mixture of these disciplines are also common.

We can notice that none of the preceding queueing disciplines depends on current time. Others, however, do exhibit time dependence with scheduling based on time-dependent **priorities** | KLE 76 |. In general, for an arriving message, the queueing discipline determines its relative position in the queue. But we can also associate with a message a numerically valued priority function of current time. Then, when the service facility becomes free, it chooses that message in the queue with the highest **instantaneous** priority. As a consequence, the order of the messages may change at any time, even between message arrivals.

Moreover, we need to consider systems that are not work conservative. Queueing theory, on the other hand, has dealt primarily with conservative systems (| KLE 76 |), i.e. no work (service requirement) is created or destroyed within the system. For example a creation of work might correspond to a "server" standing idle in the face of a non empty queue (in LYNX a message can wait before transmission even if the channel is free ,cf. 3.2.2.). Destruction of work would occur if a message were to leave before completing its "service" (in LYNX, a message can be rejected because of promptness control, cf. 4.3.).

For all of these disciplines, we need to consider preemptive priority queueing systems (a message in the process of being served can be removed from service when a message with a higher priority arrives), as well as nonpreemptive systems. For most of these queueing disciplines, we know an approach for calculating average waiting times, mean numbers of messages from each group j which are present in the system, etc. For real-time applications, however, such results fail to convey how well a given discipline performs with respect to promptness requirements. What are called for instead are more complete probabilistic descriptions of system behavior e.g., the probability distribution function (PDF) of waiting time. Such measures, in turn, call for innovative

models that permit feasible solutions via analysis or simulation. Based on results described in the section that follows, one approach we plan to pursue is the use of generalized stochastic Petri nets (e.g., stochastic activity networks | MOV84 |, | MEY85 |) to model complex queueing disciplines.

4.3. Promptness control

4.3.1. Definition

As informally discussed in Section 3.2.3, promptness control is a means for insuring that messages received at some specified destination are received on time (before or at their deadlines). This is achieved by rejecting any message that has already missed its system deadline or that will predictably miss its system deadline by virtuel of missing some intermediate deadline (determined by the system deadline). Use of promptness control is based on the policy that, in real-time applications, it is better to deny final receipt of a message, via rejection, than to deliver a message after its deadline .

More precisely, promptness control is executed at various locations in the system, called **promptness control points** (pc points) ; at a given pc point, execution takes place at different **promptness control times** (pc times) determined in a specified manner. When a message m arrives at a pc point PC_i, we presume it carries a deadline DL(m,i) for the occurrence of some future event involving m. Such a deadline may refer to a relatively immediate event, e.g., departure of m from PC(i) or may, at the other extreme, refer to the receipt of m at its final destination (in which case DL(m,i) would be the system due date for m). Relative to LYNX, for example, if PC(i) were located at the head of the scheduled queue SQ(S) (see Figure 3.1) then DL(m,i) is T(m), the deadline for transmitting m on the channels.

In general, for a message m that arrives at pc point PC(i), promptness control is executed as follows :

> While m remains at PC(i), for each successive pc time t of PC(i) that occurs after the arrival of m, PC(i) checks the condition
>
> $$t > DL(m,i).$$

If this condition is true then m is rejected.

Thus when a message m departs a pc point, it either departs normally due to an external event or departs via rejection at the first pc time t such that $t > DL(m,i)$. Note that the definition of pc execution permits a message to pass through a pc point without undergoing a "promptness check" (the comparison of current time with the message's deadline), i.e., the situation where no pc time t

occurs while the message is at the pc point. This can be avoided, however, if the pc point is designed such that the pc times include the times that messages arrive at the pc point. In this case, a promptness check is always made the instant a message arrives and, hence, every message passing through the point is checked at least once.

In addition to the stated goal of preventing the receipt of late messages, promptness control has a second important avantage when used at intermediate points in the system. Namely, by purging the system of useless late messages, system resources (buffers, processors, channels) are released for occupancy by messages which have not as yet missed their deadlines. This suggests that promptness control can improve the performance of a real-time system relative to messages that are not rejected. The crucial question, however, is whether overall performance (as quantified by the measures defined in 4.1) is improved through the use of promptness control and, if so, to what extent. The analytic study reported in the subsections that follow is an initial attempt to resolve this important question.

4.3.2. Queues with promptness control

To examine promptness control, both in general terms and in the context of the proposed system, we consider a simple queueing system, i.e., a single buffer (queue) along with a single server.

We assume that the buffer capacity is either infinite or some finite value k-1. Messages (customers) arrive as a Poisson process with rate α ($\alpha > 0$) and service time is exponentially distributed with parameter μ (the mean service rate). In other words (before promptness control), if the buffer capacity is ∞ the system is an M/M/1 queue ; if the buffer has capacity K-1 then the system is an M/M/1/K queue.

Let us now consider the incorporation of promptness control in the system by the addition of promptness control points at the outputs of the buffer and the server, i.e., the situation depicted in Figure 4.2.

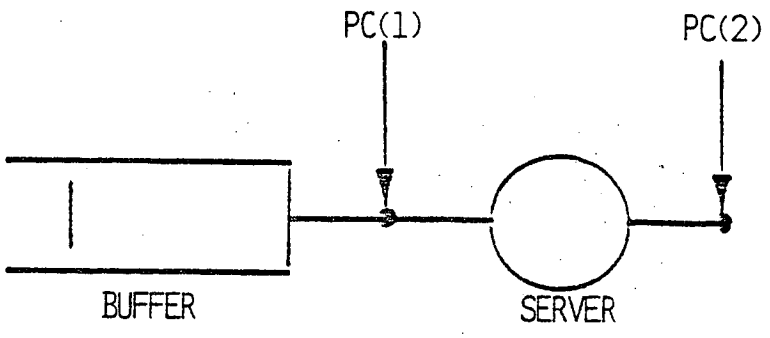

Figure 4.2

A message m that arrives at time t(m) carries a deadline DL(m) for departing the system (completing service). We assume that this is the deadline for both pc points (i.e. DL(m,1) = DL(m,2) = DL(m)). The pc times for PC(1) are the times that messages enter the server ; the pc times for PC(2) are the times that messages depart the server. Note that, when the system is nonempty, these times coincide. PC(2) insures that messages which depart the system meet their deadlines since a message m will pass PC(2) if and only if it is served and service is completed before or at time DL(m). PC(1) is an intermediate pc point whose purpose is to improve performance relative to a system without PC(1). To evaluate the performance of this system, it is necessary to describe, in greater detail, the nature of the deadlines DL(m). To this end we consider the difference between the deadline and arrival time, i.e.

$$D(m) = DL(m) - t(m).$$

D(m) is referred to as the **allowed delay** (maximum response time) of m. We permit different messages to have different allowed delays, but we assume that the number a different allowed delays is finite. In other words if we classify messages according to the value of D(m), there are finitely many message classes. More specifically we let the set

$$\{1,2,\ldots,N\} \qquad (N \geqslant 1)$$

denote the message classes and, for every message in class j we let

$$d(j) \; ; \; d(j) \in R, \; d(j) > 0$$

denote its allowed delay. (In practice, we anticipate that only a few message classes will have to be considered). Finally, we assume that an arriving message is in class j with a specified probability c(j), where

$$\sum_{j=1}^{N} c(j) = 1$$

In other words, messages of type j arrive as a Poisson process with rate c(j)· , where is the message arrival note.

Under the above assumptions, we can define the following random variables for the system in question

D = allowed delay of an incoming message

W(1) = time (relative to arrival) for a message to reach PC(1)

W(2) = time (relative to arrival) for a message to reach PC(2)

Note that the probabilistic nature of D has already been specified, i.e., for j =1,2,...,N

$P(D = d(j)) = c(j)$

As for W(1), if a message fails to reach PC(1) (this can occur in the case of a finite capacity queue where messages are rejected on arrival whenever the queue is full), we take the value of W(1) to be ∞. Similarly, if a message fails to reach PC(2) because it fails to reach PC(1) or is rejected at PC(1), we let $W(2) = \infty$.

In terms of the above variables, we now define what we refer to as **promptness control variables**, i.e., the random variables,

$L(1) = W(1) - D$

$L(2) = W(2) - D$

related to PC(1) and PC(2), respectively. Note that these variables express "lateness" (see Section 4.1) relative to a particular pc point. The importance of of L(1) and L(2) is due to the following observations. By the definition of promptness control (see 4.3.1), we know that a message m is rejected at PC(i) (i = 1,2) if and only if

$t > DL(m)$

where t (in this case) is the time of arrival at PC(i). Hence m is rejected at PC(i) if and only if

$t - t(m) > DL(m) - t(m)$,

i.e.,

$t - t(m) > D(m)$

But t - t(m) is the value of the variable W(i) provided m reaches PC(i) (i.e., $W(i) < \infty$) and D(m) is the value of D. Hence a message is rejected at PC(i) iff

$D < W(i) < \infty$

or equivalently

$0 < W(i) - D < \infty$

Hence, by the definition of the pc variables L(i),

A message is rejected at PC(i) (i = 1,2) iff $0 < L(i) < \infty$

This says, in turn, that

A message passes PC(i) iff $L(i) \leqslant 0$

Accordingly, the pc variables L(i) yield the type of information we seek, provided we can obtain certain values of their probability distribution functions. In particular consider the events

ENT = message not rejected before entering (finite queues only)

PASS1 = message passes PC(1)

PASS2 = message passes PC(2)

Then

Prob(ENT) = $P(L(1) < \infty)$

Prob(PASS1) = $P(L(1) \leqslant 0)$

Prob(PASS2) = $P(L(2) \leqslant 0)$

Note that the third probability is also the probability that an arriving message will be processed on time (before or at its deadline). Hence

NRTh = $P(L(2) \leqslant 0)$

We also have the means for determining, among all messages successfully processed, the probability distribution of the difference between allowed delay and the actual delay. As in Section 4.1.2, let E denote how early the processing was completed relative to the allowed delay, i.e. $E = - L(2)$.

Then we seek to determine the conditional distribution

$P(E \leqslant x \mid L(2) \leqslant 0)$

or equivalently

$$\frac{P(-x \leqslant L(2) \leqslant 0)}{P(L(2) \leqslant 0).}$$

Let us now consider a model that permits us to solve for these probabilities.

4.3.3. Infinite queues

We consider first the case of an infinite queue, i.e., without promptness control, the system is an elementary M/M/1 queue represented by the birth-death process given in Figure 4.3.

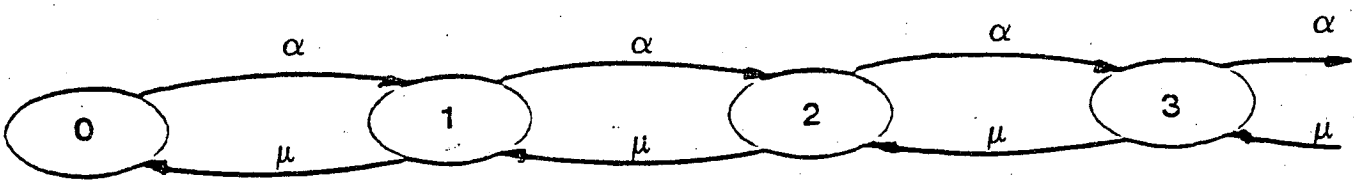

Figure 4.3

Here, the state of the system is the number of messages in the system (either in the queue or in service). However, if we now add promptness control (PC(1) and PC(2)), certain messages will not be serviced, namely, those rejected at PC(1). (Note that PC(2) does not affect the internal state behavior of the system ; PC(2) simply alters what we regard as acceptable output). Such rejections may be thought of messages which aquire service but are then served in zero time and leave the system by another exit. Under these conditions, maintaining the above notion of system state, the corresponding stochastic process is no longer Markovian.

Suppose, however, that we refine the definition of system state as follows. We number the stages of the buffer, as indicated in Figure 4.4. and classify messages according to a prediction of whether or not they will pass PC(1) after reaching PC(1). More precisely letting q denote the state of the system, if the system is empty then q = 0.

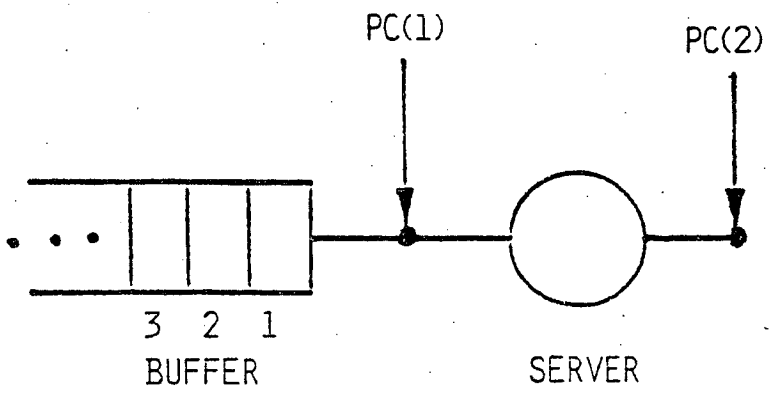

Figure 4.4

If there are m messages in the system ($m \geq 1$) then the state q of the system is the m-tuple

$$q = (q(m-1), q(m-2),\ldots,q(0))$$

where $q(0) = 1$ (indicating a busy server) and, for $k = 1,2,\ldots,m-1$,

$$q(k) = \begin{cases} 1 \text{ if message in stage k is predicted to pass PC(1)} \\ 0 \text{ else (message in stage k is predicted to be rejected at PC(1))} \end{cases}$$

where the prediction is done when the message arrives. Note that the state of the model, as defined above, represents predictions of what will happen to messages at PC(1) but does not say what actually happens to individual messages. In the long run (i.e., in steady-state), however, we find that model's probabilistic behavior does accurately reflect the actual behavior. This correspondence has been validated by simulating the actual system's behavior and comparing the simulation results with those produced by the model.

To demonstrate that the state behavior of this model is indeed Markovian, we note first that state transitions occur either on the arrival of a message or on the completion of the service of a message. In the latter case, there are transitions with rate μ (the service rate) to a uniquely determined next state. For example, suppose the current state is q = (1,0,1,1), i.e., there is a message being served, the message in buffer stage 1 will receive service (will pass PC(1)), the message in stage 2 will not and the message in stage 3 will. Accordingly, on completion of message service, the next state of the system is q' = (1,0,1) since the message that was in stage 1 passes P1 and receives service ; subsequent messages move up one stage in the buffer. As a second example, suppose now that q' is the current state. Then, on completion of service, the message in stage 1 reaches PC(1) and is rejected (instantaneously) by P(1). Hence, the message that was in stage 2 reaches PC(1), passes PC(1), and goes in service. Accordingly the next state of the system is q" = (1).

Regarding transitions that occur on message arrivals, here it is necessary to predict whether or not an arriving message will pass PC(1) (when it eventually reaches PC(1). This depends on the type of arrival (according to allowed delay), the state of the system (when a message arrives), and the value of the pc variable L(1) (see Section 4.3.2). Accordingly let Q denote the random variable whose value is the state of the system under steady-state conditions. Because of the purely random nature of arrivals (a Poisson stream), it is important to note that Q can also be interpretated as the state of the system at the time of an arrival. The latter interpretation is used in the discussion that immediately follows. (When it comes to computing the probability distribution of Q, we will use the former interpretation). For a state q and an arrival of type j ($1 < j < N$), let a(q,j) denote the probability that an arriving message will pass PC(1), given the message is type j and the state (upon entering) is q ; let $b(q,j) = 1 - a(q,j)$, the probability that an arriving task does not pass PC(1) under these conditions. Then transitions rates from state q (due to arrivals) are as depicted in Figure 4.5.

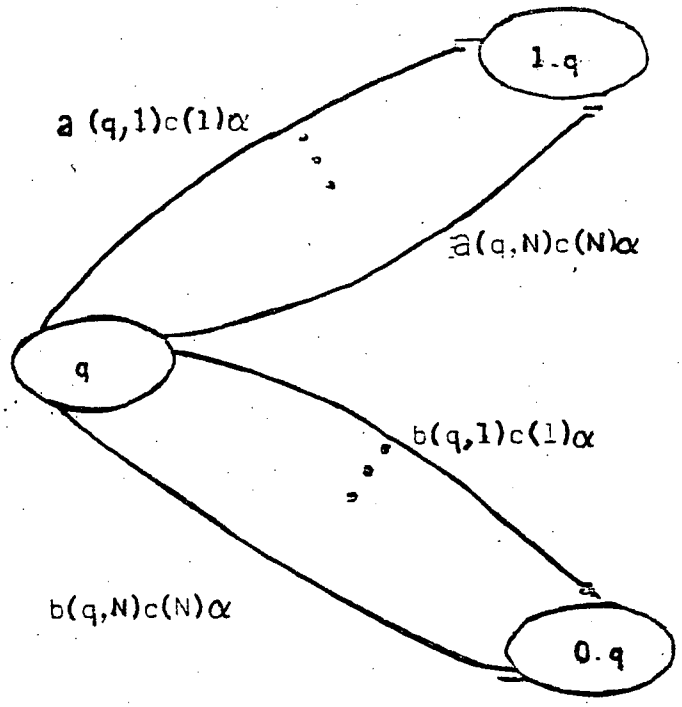

Figure 4.5

Moreover, the probabilities a(q,j) (and hence b(q,j)) can be formulated as follows . By definition,

$a(q,j) = P(L(1) \leqslant 0 \mid Q = q, D = d(j))$

Hence,

$a(q,j) = P(W(1) \leqslant D \mid Q = q, \quad D = d(j)) = P(W(1) \leqslant d(j)) \mid Q = q)$

Let M(q) = number of messages in the system that have passed or will predictably pass PC(1) when the state is q

i.e., if $q = (q(m-1),\ldots,q(0))$ then

$$M(q) = \sum_{i=0}^{m-1} q(i)$$

Then the random variable W(1), when conditioned on Q=q, depends only on M(q).

More precisely, in the special case where M(q) = 0, which occurs only when q = 0 (no messages in the system), W(1) = 0 with probability 1. If M(q) > 0 then, because individual service time are exponentially distributed with rate μ, W(1) (conditioned by Q = q) is distributed as an Erlang-k distribution (see [CIN75], p. 83, for example) where k = M(q)

More precisely, if we let Erl(k,t) denote this distribution, i.e.,

$$\text{Erl}(k,t) = 1 - e^{-\mu t} \sum_{i=0}^{k-1} \frac{(\mu t)^i}{i!}$$

Then, by the above observations,

$$a(q,j) = \begin{cases} \text{Erl}(M(q),d(j)) \text{ if } M(q) > 0 \\ 1 \text{ if } M(q) = 0 \end{cases}$$

$$b(q,j) = 1 - a(q,j)$$

Finally, if we let a(q) and b(q) denote the total transition rate from state q to state (1,q) and (0,q), respectively (see fig. 4.5.), it follows that

$$a(q) = \begin{cases} \sum_{j=1}^{N} c(j)\, \text{Erl}(M(q),d(j)) \text{ if } M(q) > 0 \\ 1 \text{ if } M(q) = 0 \end{cases}$$

$$b(q) = 1 - a(q)$$

The above arguments establish that the process in question is a Markov process and, moreover, provide the specific details of the process structure. On closer examination, due to the infinite capacity of the queue we find that all states q having the same value of M(q) behave identically (relative to other such sets of states). More precisely, if we lump the states according to the function M and let $\bar{Q}$ denote the new state variable then $\bar{Q}$ takes values in the set

0,1,2,...

and the corresponding process is likewise a Markov process.

A state k of this lumped process represents all the states q (of the original process) such that

M(q) = k

In other words (by the definition of M(q)), k is the number of messages in the system that have passed or will predictably pass PC(1). Accordingly, if we define a(k) to be the value a(q) (see above) for any q such that M(q) = k,

$$a(k) = \begin{cases} \sum_{j=1}^{N} . c(j)\, Er1(k, d(j)) \text{ if } k > 0 \\ 1 \text{ if } k = 0 \end{cases}$$

Moreover, since $a(q)\,\alpha$ is the transition rate from state q to state (q,1) and α is the total rate from q due to arrivals, the value a(k) has the following interpretation :

a(k) = probability that an arriving message will (eventually) pass PC(1), given the system is in state k at the time of arrival.

The resulting (lumped) process is a birth-death process having the state-transition-rate diagram given in figure 4.8.

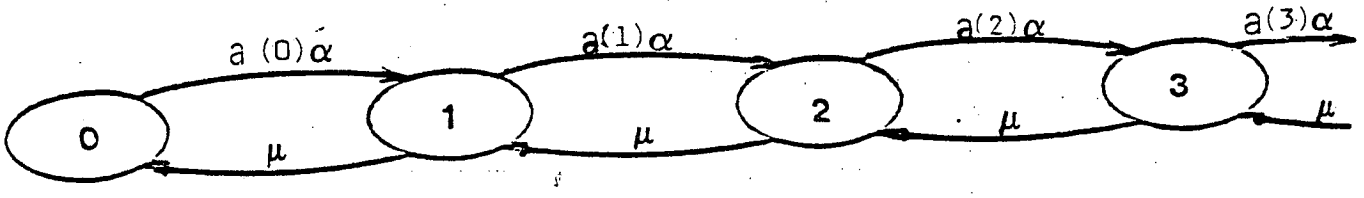

Figure 4.6

If we let

p(k) (k = 0, 1, 2,...)

denote the steady-state probability of occupying state k, solution of the p(k) is well known for birth-death processes (see | CIN75 |, for example). In terms of the p(k), the performance measure (see 4.1.2).

NRTh = probability (under steady-state conditions) that a submitted message is received on time

can be formulated as follows.

We note first, as indicated earlier in this section, that NRTh is equal to the probability that the lateness at PC(2) is less than or equal to 0, i.e.

NRTh = Prob (L(2) $\leqslant$ 0).

To determine this probability, we condition the event $L(2) \leqslant 0$ on the state of the system at the time the message is submitted and on the type of message (according to allowed delay) being submitted. In other words, given that the state is k and the message type is j, we consider the probability

$$\text{Prob}\,(L(2) \leqslant 0 \mid \bar{Q} = k,\ D = d(j))$$

Equivalently, in terms of W(2) (the time, relative to submission , for a message to reach PC(2)), we can express this probability as

$$\text{Prob}\,(W(2) \leqslant d(j) \mid \bar{Q} = k))$$

Moreover, $\bar{Q} = k$ says that there are k messages in the system that have either passed PC(1) or will predictably pass PC(1). Hence, there are k messages in the system that are predicted to be served (one of them is in service if $k > 0$) before the incoming message reaches PC(1).Since $W(2) \leqslant d_j$, we know that the incoming message passes PC(1) and thus, will also be served. Consequently, the time W(2) for this message to reach PC(2) is the sum of k + 1 exponentially distributed service times, i.e.,

$$\text{Prob}(L(2) \leqslant 0 \mid \bar{Q} = k,\ D = d(j)) = \text{Erl}\,(k + 1,\ d(j))$$

Summing over message types and recalling that $c(j) = \text{Prob}\,(D = d(j))$,

$$\text{Prob}(L(2) \leqslant 0 \mid \bar{Q} = k) = \sum_{j=1}^{N} c(j)\ \text{Erl}\,(k+1,\ d(j))$$

In turn, summing over the states where, as introduced above, $p(k) = \text{Prob}\,(\bar{Q} = k)$, we conclude that

$$\text{NRTh} = \sum_{k=0}^{\infty} p(k) \left(\sum_{j=1}^{N} c(j)\text{Erl}\,(k+1,\ d(j)) \right)$$

Actual calculations of NRTh, using this formula, would necessitate a means of calculating the p(k) as an approximation obtained by truncating the sum at some appropriate point (the values of both p(k) and Erl (k+1, d(j)) decrease with increasing k).

However, our interest in the infinite queue case is primarily to set the stage for the discussion of finite queues with promptness control. As we will see in the subsection that follows, much of what we have already said applies to finite queues as well. However, it is no longer possible to do the type of state lumping we did in the infinite case.

4.3.4. Finite queues

When the buffer is finite, an analysis of the type presented above is complicated by the fact that messages can be rejected upon arrival, due to a full queue. Moreover, it is important to consider this more complex case in the context of LYNX, since we anticipate that SQ(S) (see Figure 3.1) will have a relatively small capacity in the order of 5-10 stages. Thus, only under very hightly loaded conditions could we regard SQ(S) as having infinite capacity.

Suppose then that the system is an M/M/1/K queue (i.e., the buffer has capacity K-1) augmented by two promptness control points PC(1) and PC(2) as shown in Figure 4.4. Using the same representation of states employed in the previous subsection, since the number of messages m in the system is at most K, the state set will consist of the state q=0 (empty system) along with all m-tuples of the form

$$q = (q(m-1),q(m-2),\ldots,q(0))$$

such that $1 \leqslant m \leqslant K$ and q(0)=1. Hence the total number of states is

$$1 + \sum_{m=1}^{K} 2^{(m-1)} = 2^K$$

Transition rates from states are determined in the manner described for infinite queues (prior to lumping), except for states representing a full queue. In the above notation, a full queue is represented by a K-tuple and there are 2^{K-1} states in this category. For such states a state transition can occur only when service (for the message currently be served). is completed. Thus the transition rate from a full queue state is μ , with no contribution due to message arrivals. As in the infinite queue case, the next state resulting from a service completion is unique and is determined in a manner identical to that described in the previous subsection.

The resulting stochastic process is again a Markov process and its state-transition-rate diagram can be completely specified for a given value of K. The resulting diagram for the case K=3 (a 2-stage buffer) is shown in Figure 4.7.

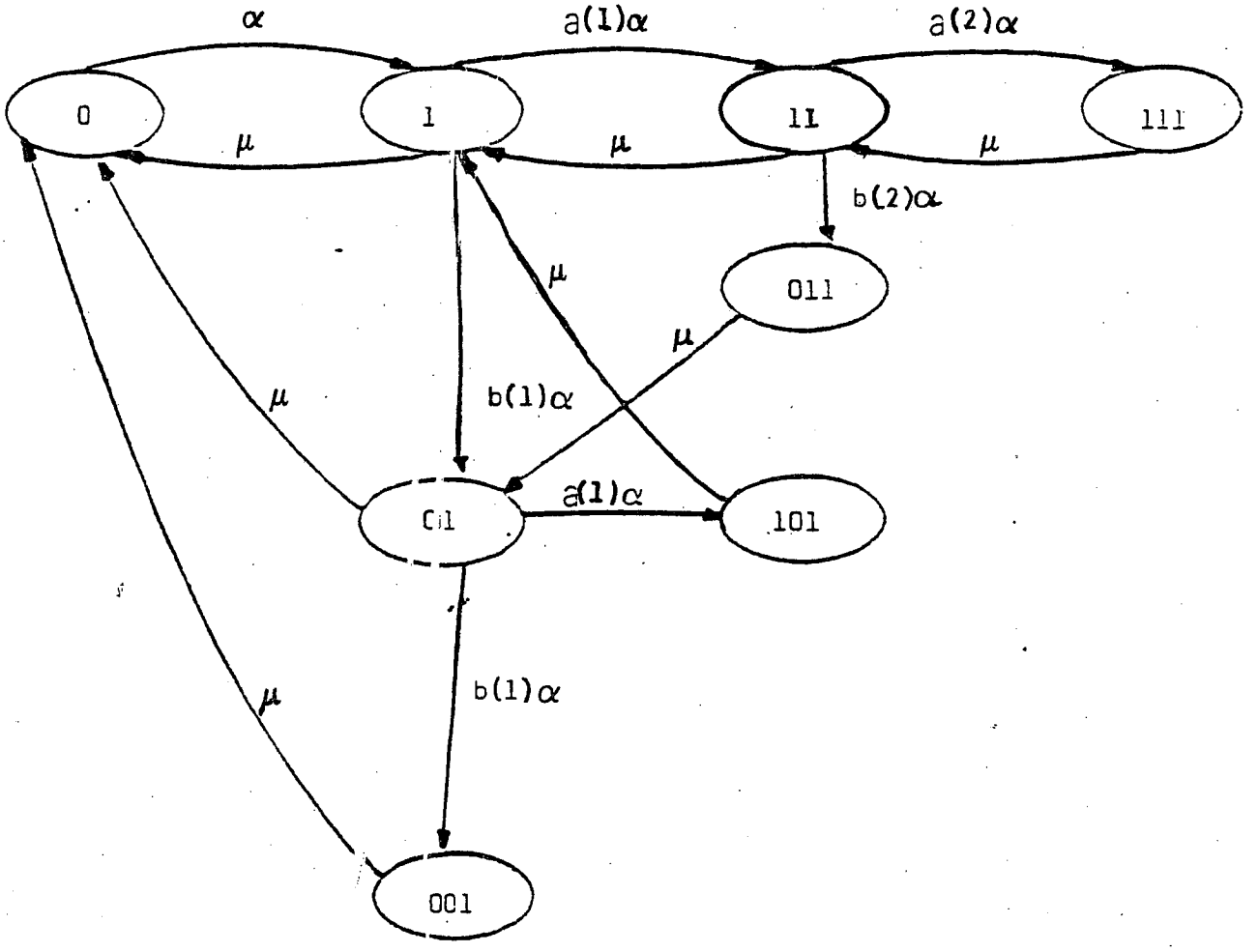

Figure 4.7

States are named by 0-1 sequences which abbreviate the m-tuples described above (i.e., parentheses and commas are deleted from the m-tuples). Note that the diagram is drawn so that states with a common value of M(q) are aligned vertically, beginning with M(q)=0 at the left of the diagram and ending with M(q)=K at the right. Transition rates due to arrivals are specified in terms of coefficients a(k) and b(k) that were formulated for the lumped process in Section 4.3.3. Note, however, that states here are not being lumped ; a(k) is simply an abbreviation of a(q), presuming that M(q)=k (a similar statement applies to b(k)).

Unlike what we observed in the infinite queue case, the model derived above for finite queues cannot be simplified via state lumping. More precisely, one can show that any reduction of the state space, via lumping, results in a stochastic process that is no longer Markovian. On the other hand, the state space appears to have a manageable size (e.g., if K=10 there are 1024 states), particularly if the model is constructed by alternative means. One such approach is the use of stochastic Petri nets, as illustrated in the subsection that follows.

Given this model, with parameters K (system capacity), N (number of message types), d(1),...d(N) (allowed delay distribution), α (arrival rate), and μ (service rate), the performance measure NRTh can be solved in a manner similar to that discussed for infinite queues. Again, differences that exist are due mainly to the presence of full queue states. As earlier we let Q be the random variable

Q = state of the system (stochastic process) at the time of an arrival

and let M be the function, defined on values q of Q, where

M(q) = number of messages in the system that have passed or will predictably pass PC(1) when the state is q

to account for full queue states, we introduce a second function TM, again defined on the values of Q, where

TM(q) = total number of messages in the system when the state is q.

Thus, by our representation of states as m-tuples, it is immediate that

$$TM(q) = \begin{cases} m \text{ if } q \text{ is a m-tuple and } q \neq 0 \\ 0 \text{ if } q=0. \end{cases}$$

Accordingly,

TM(q) = K iff the queue is full in state q.

As M and TM are functions of the random variable, Q (the state variable), M(Q) and TM(Q) are likewise random variables which, with only a slight risk of ambiguity, are abbreviated as M and TM, respectively.

To determine NRTh = Prob(L(2) $\leqslant$ 0), we condition the event L(2) $\leqslant$ 0 on the variables. D (allowed delay of an arriving message), M and TM, i.e., we consider the conditional probability

$$\text{Prob}(L(2) \leqslant 0 \mid D = d(j),\ M = k,\ TM < K)$$

where K (as earlier) is the message capacity of the system. Alternatively (compare with the infinite queue case), this probability can be expressed in terms of W(2) as

$$\text{Prob}(W(2) \leqslant d(j) \mid M = k,\ TM < K)$$

Since TM < K, an arriving message will enter the system. Moreover, as in the infinite queue case, W(2) < d(j) implies that the message passes PC(1). Consequently (see the infinite queue case for details),

$$\text{Prob}(L(2) \leqslant 0 \mid D = d(j),\ M = k,\ TM < K) = \text{Erl}(k + 1,\ d(j)).$$

Summing over message types, we have

$$\text{Prob}(L(2) \leqslant 0 \mid M = k,\ TM < K) = \sum_{j=1}^{N} c(j)\ \text{Erl}(k + 1,\ d(j)),$$

and it remains to formulate the joint probabilities of the events M=k and TM < K.

In the latter regard, for each state q (an m-tuple) let

$$p(q) = \text{Prob}(Q = q)$$

where Q is, for the moment, interpreted as the state of the process under steady-state conditions. With this interpretation, values of p(q) for each state q can be solved for by classical finite-state Markov process solution technique (see | CIN75 | for example). Returning to our former (and equivalent) interpretation of Q, i.e., the state of the system (under steady-state conditions) the time of an arrival, we can now formulate the desired probabilities. More precisely, since M = M(Q) and TM = TM(Q), it follows that

$$\text{Prob}(M = k,\ TM < K) = \sum_{\substack{M(q)=k,\\ TM(q)<K}} p(q)$$

If we let r(k) denote this probability ($0 \leq k < K$), next we observe further that r(k) vanishes in the case where k=K, i.e., r(K) = 0. This results from the fact that, by definition, M(q) < TM(q) and, hence, the joint event M=K, TM < K is impossible. Combining the above derivations and observations, we obtain the formula we seek for the normalized real-time throughput, i.e.,

$$NRTh = \sum_{k=0}^{k-1} r(k) \left(\sum_{j=1}^{N} c(j)\ \text{Erl}(k+1, d(j)) \right)$$

Alternatively, by changing the order of summation,

$$NRTh = \sum_{j=1}^{N} c(j) \left(\sum_{k=0}^{K-1} r(k)\, Erl(k+1, d(j)) \right)$$

The latter is interesting since the sum on the right expresses NRTh as conditioned by message type (allowed delay), i.e.,

$$Prob(L(2) \leqslant 0 \mid D = d(j)) = \sum_{k=0}^{K-1} r(k)\, Erl(k+1, d(j)).$$

Application of the modeling and solution methods described above is illustrated in the subsection that follows.

4.3.5. Application

The previous two subsections have described an analytic method which can be used to evaluate queueing systems with promptness control. In this subsection, we examine an application of this method for the case of a finite queue. In this case, the method described in the previous subsection permits the state behavior of the system to be represented by a finite-state Markov process. This Markov process could then be solved, using standard Markovian solution techniques, to determine the performance of the system.

Use of such solution techniques, however, necessitates the determination of the state-transition-rate diagram (or, alternatively, the infinitesimal generator matrix) of the Markov process. Moreover, if the capacity of the queue is even moderately large, this can be both complicated and tedious. One way to alleviate such complictation is to use a model which can represent the system in a natural manner and whose state behavior constitues the desired Markov process. If such a model is sufficiently formal, a computer program may be employed to determine the state-transition-rate diagram, hence relieving the burden and possible inaccuracies of a manual derivation.

Accordingly, in the application that follows, we use stochastic Petri nets (SPNs) (see | BEY81 | for example) to model the system and employ RdPS, a computer program developed at the CNAM, to aid evaluation of the modeled system. Given an SPN model with a Markovian state behavior, this program is able to automatically generate the state-transition-rate diagram of the Markov process representing the state behavior of the model. Moreover, it incorporates some standard Markovian solution techniques which determine the steady-state probability distribution of the (reachable) states of the model.

Subsequent to the work reported here, a more general class of **networks**, referred to as **stochastic activity networks** (SANs), has been developed as a modeling base for evaluating the performability of distributed real-time systems | MOV84 |, | MEY85 |, | MOV85 |. In particular, SANs have been used to model more complex versions of the example described in this section | MOV85 |.

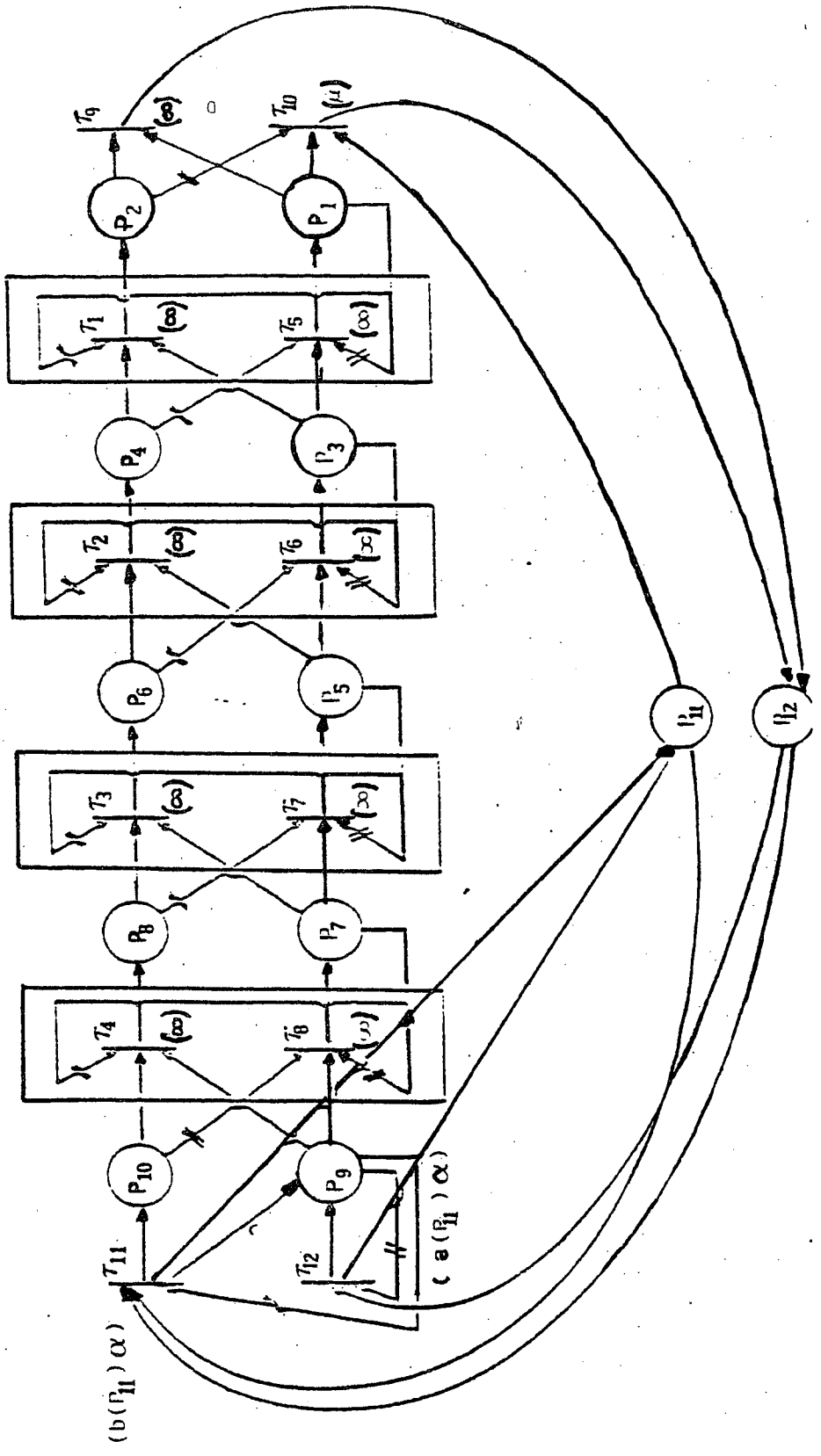
T9
(8)
T10
(4)
P2
P1
T1
(8)
T5
(∞)
P4
P3
T2
(8)
T6
(∞)
P6
P5
T3
(8)
T7
(∞)
P8
P7
T4
(8)
T8
(∞)
P10
P9
T11
T12
P11
P12
(b(P11)α)
(a(P11)α)

Figure 4.8

Figure 4.8 is an example of an SPN model which represents a single-server queue with promptness control (see the discussion of the previous subsection). The capacity in this example is 5 (server plus a queue with capacity 4). In this model, firing of transtion T_{11} or T_{12} represents the arrival of an incoming message. Firing of transition T_{12} represents the arrival of a message which will pass promptness control point (pc point) PC(1) ; firing of transition T_{11}, on the other hand, represents the arrival of a message which will eventually be rejected at pc point PC(1). The 4 blocks consisting of transitions T_1, T_2,...,T_8 function similarly ; each of these blocks represents the shifting of a waiting message in the buffer to the next empty stage. Firing of transition T_9 represents the rejection of a message which has missed its deadline according to pc point PC(1). This rejection happens whenever the channel (server) becomes free and a late message is waiting in the buffer. Places P_3, P_5, P_7 and P_9 represent the first, second, third and fourth stages in the buffer, respectively. A token in one of these places represents a message at the corresponding buffer stage. Places P_4, P_6, P_8 and P_{10} represent the status of the messages, with respect to promptness control PC(1), at the first, second, third, and fourth buffer stages respectively. A token in any of these places means that the message at the corresponding buffer stage will be rejected at pe point PC(1). Places P_1 and P_2 represent the status of the channel and the messages which arrive at pc point PC(1). Having a token in place P_1 and no token in place P_2 means that the channel is busy transmitting a message. Tokens in both places P_1 and P_2 represents a late message which should immediately be rejected at pc point PC(1). Places P_{11} and P_{12} are redundant places as far as the modeling of the system is concerned. P_{11} represents the number of the messages in system which will be transmitted (i.e., will not be rejected at pc point PC(1)). P_{12} represents the system capacity less the total number of the messages in system (either waiting in the buffer or being transmitted).

When a message arrives via firing of T_{12}, one token is put in P_9 meaning that this message will pass pc point PC(1) when it eventually arrives there. When a message arrives via firing of T_{11} one token is put in both places P_9 and P_{10}, meaning that this message will eventually be rejected at pc point PC(1). The (previously mentioned) blocks then sift this message accordingly, if necessary. This is done, more specifically, as follows. When a block has a token in both of its places (i.e., representing a message which will be rejected at pe point PC(1)) while the next block has no token in any of its places (i.e., the next stage is empty), its upper transition will immediately fire which will then remove all tokens of the places of this block and put a token in each place of the next block (i.e., the message is immediately shifted to the next stage). Similarly, when a block has a token in its lower place and no token in its upper place (i.e., representing a message which will pass pc point PC(1) while the next block has no token in any of its places, its lower transition will immediately fire which will then remove the token of the lower place of this block and put one token in the lower place of the next block (i.e., the message is immediately shifted to the next stage). When a token is put in P_1 while P_2 is empty, it means that a message is being transmitted. The end of the transmission will be represented by firing of T_{10}. When a token is put in both places P_1 and P_2, it means that a late message has arrived at pc point PC(1) and which should immediately be rejected. This rejection is represented by instantaneous firing of transition T_9. This summarizes an informal description of logical aspects of the model of Figure 4.8. Timing aspects of the model are described by the numbers which are inside parentheses in Figure 4.8 These numbers are assigned to the transitions of the model and represent the rate of the firing of these transitions.

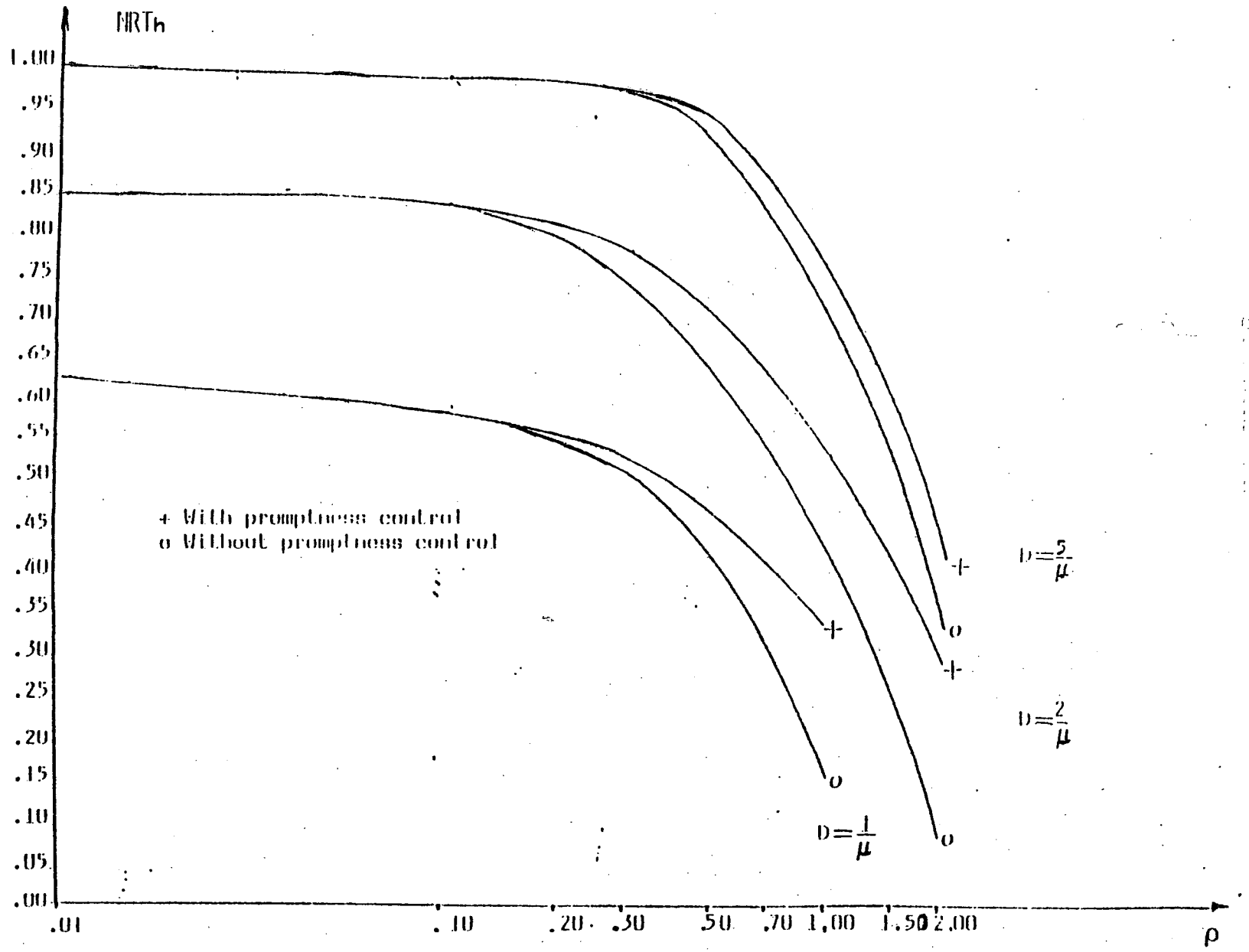
NRTh
1.00
.95
.90
.85
.80
.75
.70
.65
.60
.55
.50
.45
.40
.35
.30
.25
.20
.15
.10
.05
.00
.01
.10
.20
.30
.50
.70
1.00
1.50
2.00
ρ
+ With promptness control
o Without promptness control
D=5/μ
D=2/μ
D=1/μ

Figure 4.9

Using RdPS, the following evaluation results were derived from this model. Relative to the performance measure NRTh (see section 4.1.2), i.e., NRTh = probability (under steady-state condition) that a submitted message is received on time, Figure 4.9 displays the performance of systems with promptness control (i.e., with an intermediate pc point PC(1)) as compared to systems without promptness control. Specifically, for three different choices of allowed delay (e.g., D $=5/\mu$ means that the allowed delay is 5 times the mean service time), Figure 4.9 gives plots of NRTh as a function of the traffic intensity $\rho = \alpha\mu$ for systems with promptness control (marked by "+") and without promptness control (marked by "o"). As is shown, the performance of a system with promptness control is always superior to that of a system withotu promptness control.

The remaining results concern more refined aspects of the system where three events are considered : the event that a message enters the system (i.e., the queue is not full), denoted by ENT ; the event that a message passes pc point PC(1), denoted by PASS2. Figure 4.10 shows the probability of ENT, i.e., the probability that a message enters the system, as a function of traffic intensity ρ for various allowed delays. As is shown, this probability decreases as the allowed delay increases no matter what the traffic intensity. An informal interpretation is that as allowed delay increases there will be less rejections at pc point PC(1), which results in less room for the incoming messages which, in turn, means a lower probability of entering the system for these messages.

Figure 4.11 shows the probability of PSS1, i.e., the probability that a message passes pc point PC(1), again as a function for ρ different choices of allowed delay. As one might guess, this probability increases as the allowed delay increases. However, in both the lightly loaded (small ρ) and heavily loaded cases (large ρ), these probabilities are relatively independent of allowed delay.

Figures 4.12 and 4.13 display the probabilities of the three event ENT, PASS1, and PASS2 as a function ρ for two different allowed delays. As is shown, the probability of ENT is greater than that of PASS1 which, in turn, is greater than that of PASS2. The reason is quite straightforward ; only some of the messages that enter the system pass pc point PC(1) and, among the latter, only some are able to pass pc point PC(2). Finally, comparing Figures 4.12 and 4.13, it is obvious that these 3 probabilities have greater differences (for a given traffic intensity) as the allowed delay becomes smaller.

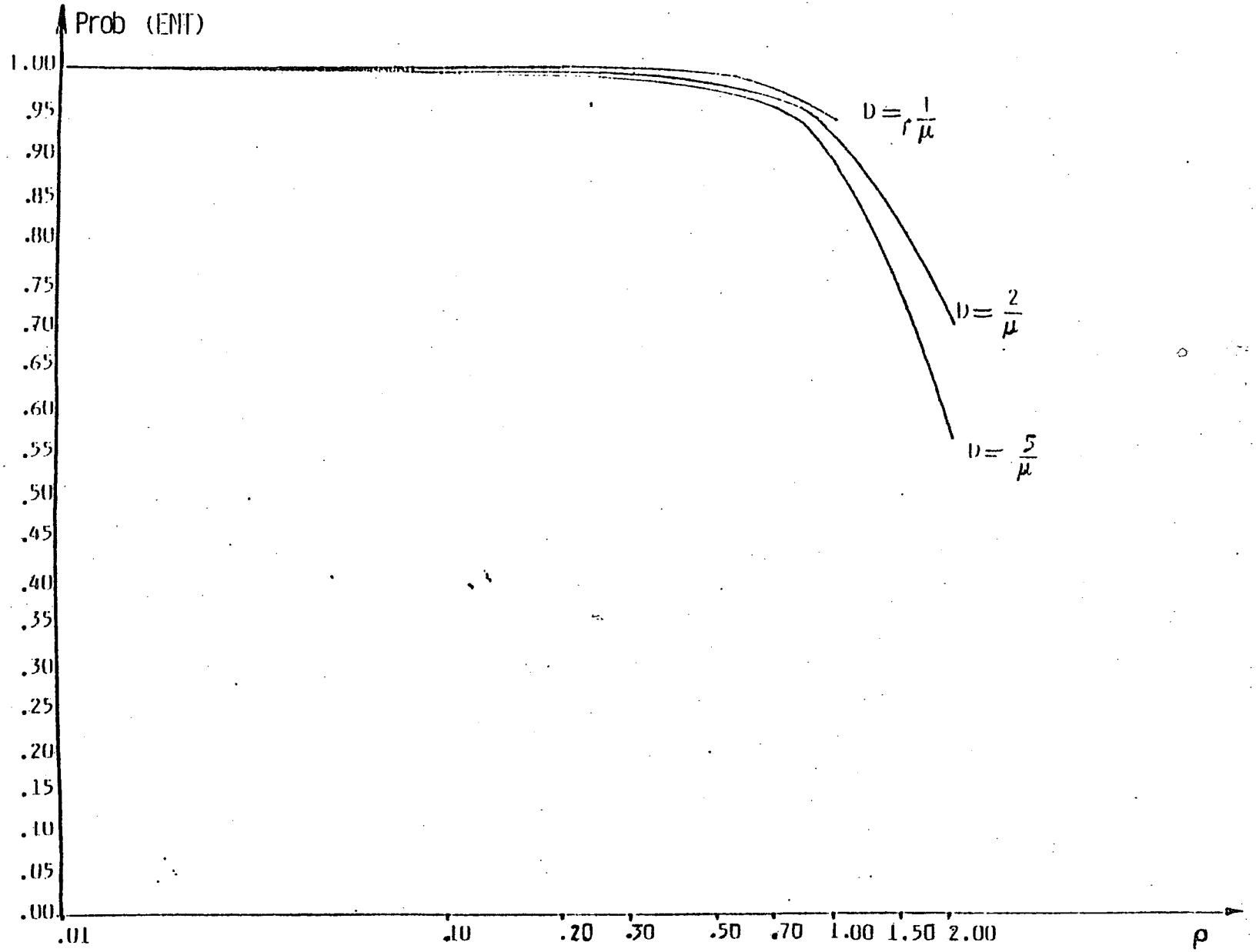
Prob (ENT)
1.00
.95
.90
.85
.80
.75
.70
.65
.60
.55
.50
.45
.40
.35
.30
.25
.20
.15
.10
.05
.00
.01
.10
.20
.30
.50
.70
1.00
1.50
2.00
ρ
D = 1/μ
D = 2/μ
D = 5/μ

Figure 4.10

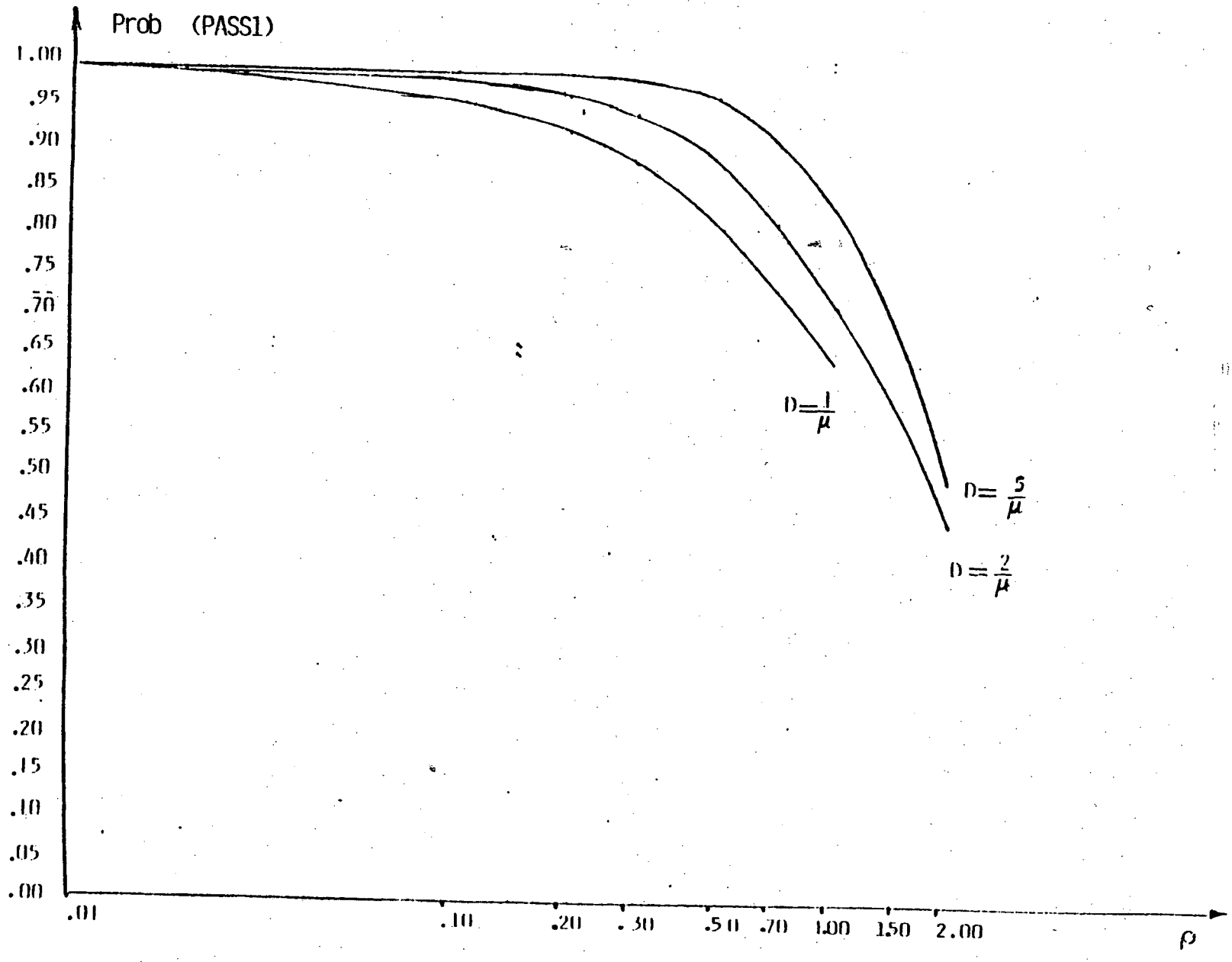
Prob (PASS1)
1.00
.95
.90
.85
.80
.75
.70
.65
.60
.55
.50
.45
.40
.35
.30
.25
.20
.15
.10
.05
.00
D=1/μ
D=5/μ
D=2/μ
.01
.10
.20
.30
.50
.70
1.00
1.50
2.00
ρ

Figure 4.11

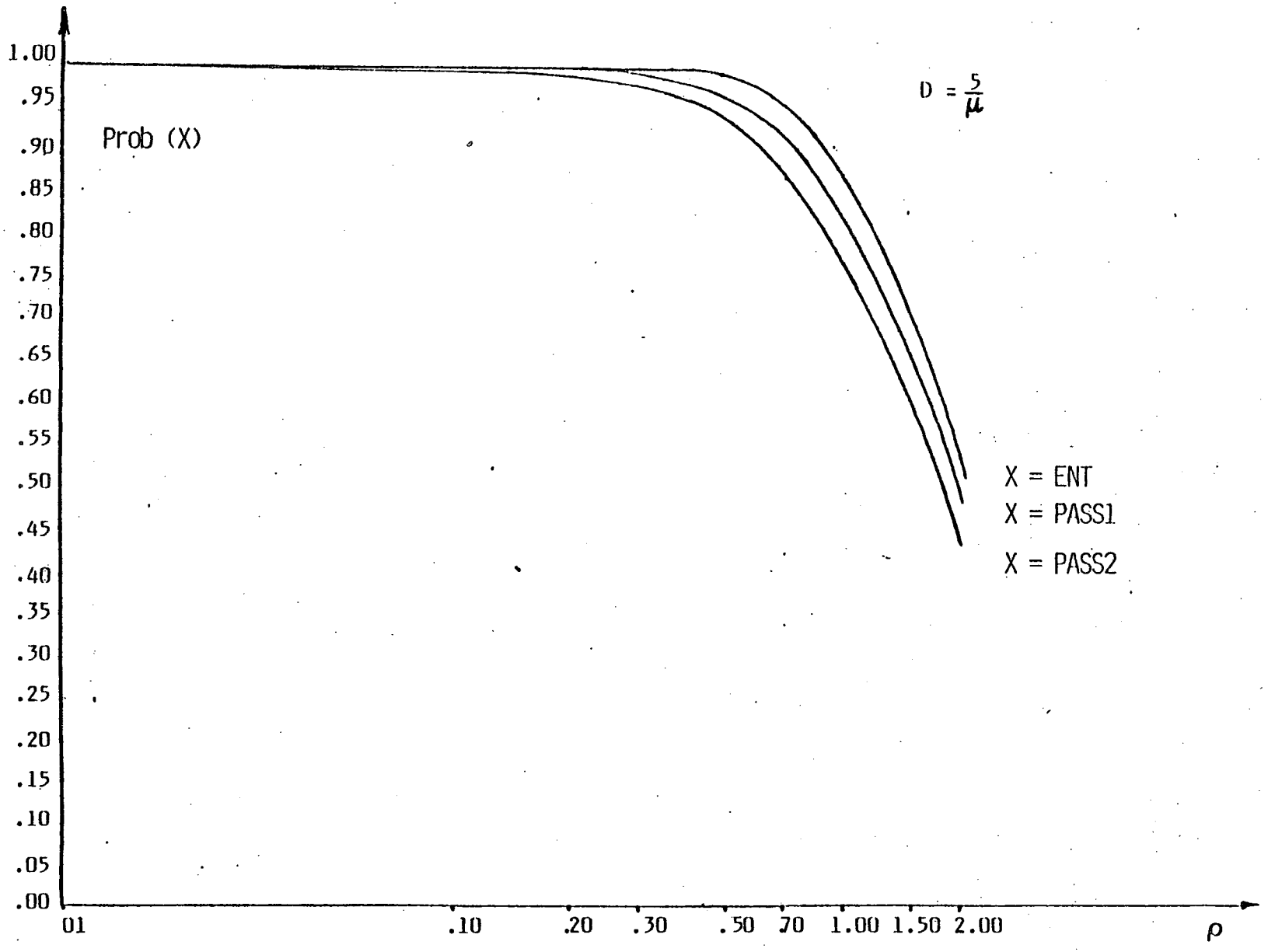
1.00
.95
.90
.85
.80
.75
.70
.65
.60
.55
.50
.45
.40
.35
.30
.25
.20
.15
.10
.05
.00
Prob (X)
D = 5/μ
X = ENT
X = PASS1
X = PASS2
01
.10
.20
.30
.50
70
1.00
1.50
2.00
ρ

Figure 4.12

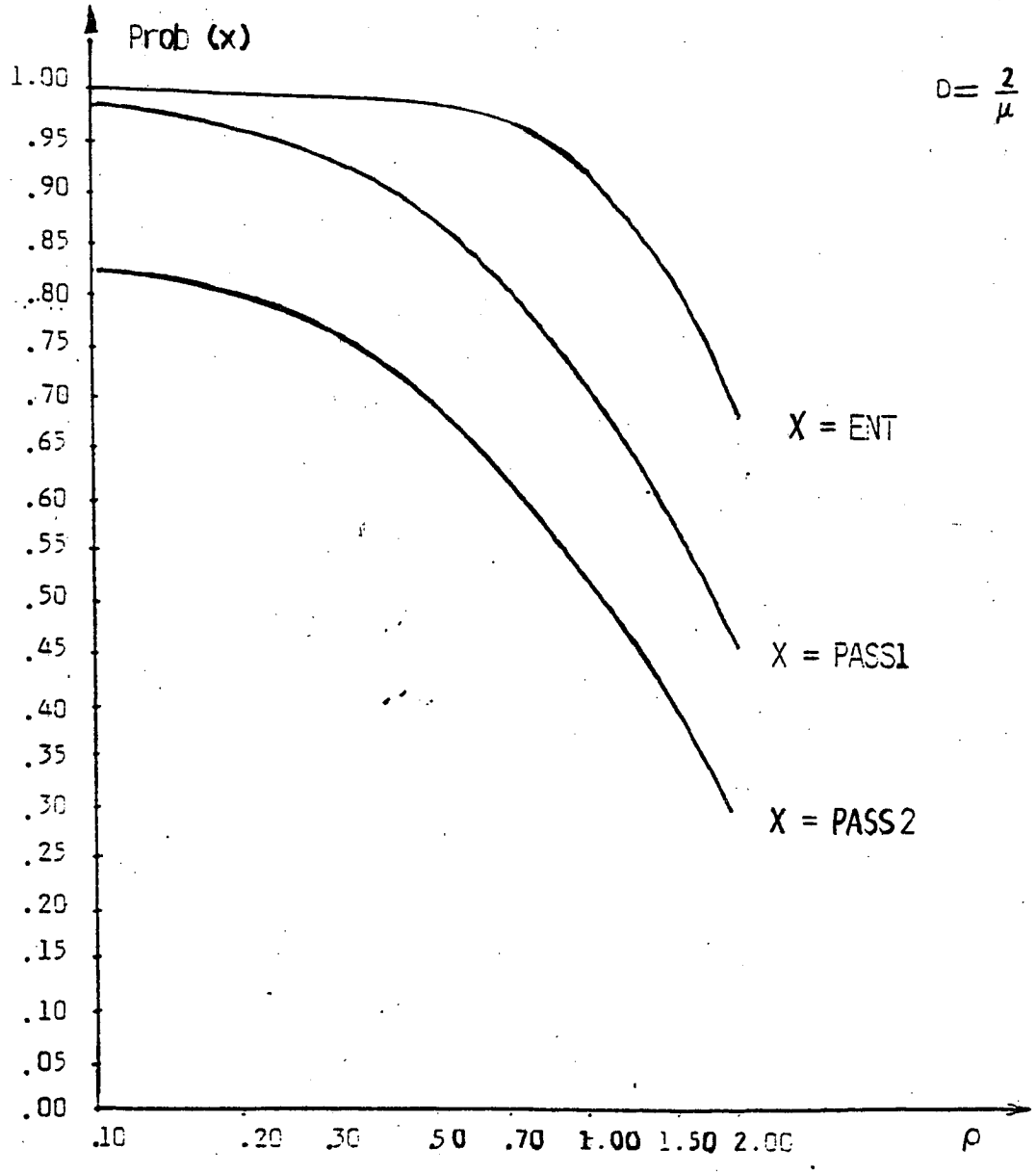
Prob (x)
1.00
.95
.90
.85
.80
.75
.70
.65
.60
.55
.50
.45
.40
.35
.30
.25
.20
.15
.10
.05
.00
D= 2/μ
X = ENT
X = PASS1
X = PASS2
.10
.20
.30
.50
.70
1.00
1.50
2.00
ρ

Figure 4.13

4.4. System-wide message scheduling

4.4.1. Introduction

Between the time t at which a SMAP gets hold of the message m ahead of its SQ(S) queue and the time at which m effective transmission starts, three sucessive scheduling mechanisms may occur :

First, at time t, a scheduling algorithm may determine a waiting period of time | t,T | so that m first transmission attempt occurs at time T. Momentarily, let us call it a latency scheduling algorithm.

At time T, SMAP may find the channel either in the idle state and succesfully transmit m or find the channel in the busy state and use a second scheduling algorithm to determine when to retry m transmission. Naturally, there may occur successive retrys before the channel is found in the idle state.

Lastly, having found the channel in the idle state, SMAP starts to transmit m but m may collide. Then SMAP uses a third scheduling algorithm which is actually the collision detection resolution .

These two last scheduling algorithm are generally termed as channel access protocols.

The present section first discusses the ability of known channel access protocols to meet real-time requirements. Then it presents our research concerning a latency scheduling algorithm which is a completely new aspect of investigation in the transmission domain and is precisely termed as a system-wide transmission first attempt scheduling algorithm.

4.4.2 Real-time channel access protocols

In this section, we discuss channel access protocols which are suited to real-time local area networks and, in particular, satisfy the objectives of our research.

We have decided to limit our scope to random access protocols because we have found these protocols very suitable to our objectives of flexibility and robustness.

As a first step, we have examined the existing literature on design and evaluation of random access protocols in order to assess their relevance to our research. The results of this investigation are summerized as follows.

1) Most existing random access protocols have not been designed for real-time applications. As a result, they are not usually suitable for real-time environments. Some limited concerns have been found in this regard, and protocols such as priority access protocols have been proposed.

2) Measures usually utilized for evaluating channel access protocols are not very appropriate for real-time access protocols. Some solutions have been found in this regard, but they are of limited scope.

3) The modeling techniques used to evaluate random access protocols are often simulation methods. Analytical methods are also used but they are usually difficult to apply. Markov renewal theory and, in some cases, queueing modeling techniques have been utilized in analytical methods.

Before starting to discuss any of these results in detail, let us make it more clear what we mean by a real-time system in general, and real-time channel access protocols in particular.

By a real-time system we mean a system which includes tasks having some specified real-time requirements: More specifically, any such task is required to be processed within some specified periods of time after it is initiated. When it fails to do so, a failure of that task will occur. These failures could be viewed as faults in the system, and their occurrances could consequently have effect on the overal properties of the system. It is this effect which distinguishes a system as real-time system. The more serious this effect is on the system properties, the more real-time the system will be.

In order to appropriately design and evaluate a real-time channel access protocol, it is important to have a specification of these systems in general. They are expected to satisfy all specifications which have been previously mentioned the about real-time systems. This would be the case if we view the messages as tasks and their processing (from the time they are sent to the channel until the time when they arrive at the destination) as processing of tasks in real-time systems. As a result, in a real-time access channel protocol, it is required that each message arrives at its destination(s) within some specified period(s) of time. When a message fails to reach its destination in time we consider this to be a failure of the message. Accordingly late messages will result in reduced performance.

In order to be more specific, we now present the specification of a real-time channel access protocol. Let us assume that each source has only one buffer which can be occupied by at most one message at any particular time. The occupation of the buffers (because of the arrival of messages) happens randomly. When a message enters a buffer it carries a deadline. Different messages have different deadlines before which it is required to arrive at its destination(s). If it fails to do so (i.e., if it misses its deadline), a failure of that message will happen. The present messages in buffers will compete among each other for accessing the channel (so that they can be transmitted to their destinations). Only one message at any time can access the channel. A real-time channel access protocol is a scheduling mechanism which resolves this competition among the present messages in buffers. Our goal is then to design scheduling policies which optimize the real-time properties (i.e., some performability) of the system. It is clear that the definition of some appropriate measure which characterizes the real-time properties of these scheduling policies is a necessary condition for an appropriate design and evaluation of these systems.

Now that we have stated some specification of the real-time systems and in particular a real-time channel access scheduling policies, we can discuss in more details the first two results of the investigation mentioned before.

1 - Most existing random access protocols are not useful for real-time LANs because they have not been designed for real-time applications. These protocols have been mainly designed for transactional environments where there is practically no real-time requirement in communication of the messages. However, recently, some concerns have been found in this regard. Franta and Bilodeau have proposed a prioritized CSMA protocol based on staggered delays | FRA80 |. The channel access protocol is suitable for environments where different nodes of the network have different priorities. Others | CHL80 |, | TOB82 |, | ONO78 |, | IID80 | have proposed message-based priority random access protocols. These protocols are appropriate for environments where there are different classes of messages with different priorities. In order to show the relevance of these protocols to our research and particularly to real-time applications, we will describe one of them, namely that of Franta and Bilodeau | FRA80 |, in more detail as follows. For this protocol, at each node i the following algorithm is performed :

Loop 1 : wait until message to transmit

Loop 2 : if carries absent

Then transmit message

if collision

Then wait D(i) time units, go to Loop 2
else go to loop

else wait until carrier absent
wait D(i) time units, go to Loop 2

$0 < D(1) < D(2) <...< D(n)$

$D(i+1) - D(i) > d$

with d representing the maximum length path propagation delay, and n the number of network nodes. D(i) values are used to prioritize the initial and subsequent transmission attempts, and they are static.

This protocol, as far as we are concerned, is a very distributed one, i.e., there is very little global information or resources shared by different nodes of the network. So it satisfies the usual properties of distributed systems, namely our objectives of flexibility and robustness. Now suppose a real-time environment where the messages of each node i of the network have real-time requirements of being at their destination(s) within a specified period of time E(i) (depending on the node i) after they have been sent for transmission. Let $0 < E(1) < E(2) <...<E(n)$. Then by choosing some suitable D(1), D(2),...,D(n) as applied in previous protocol, one could design an appropriate prioritized CSMA protocol based on staggered delays which would be suitable to our particular real-time environment.

Such a deadline oriented channel access protocol is a correct approach for real-time environments, whereas protocols in which priority is based on station identifiers network topology are out of our scope. Nevertheless, an other concern is to reduce the time during which the channel is idle because timouts are not issued. Deadline oriented channel access protocols that minimize channel idle period of time have been specified within | LEL83b |.

An other point that is stressed out in | LEL83b | is the **epoch** notion whereby, once a collision resolution is started, no other message (or only limited number of them) may enter the collision resolution and be successfully transmitted. Any **deterministic collision resolution** must observe this epoch mecanism if it is meant to guarantee an upperbounded collision resolution delay.

2 - A few common measures can be typically used to evaluate random access protocols. They are average throughput, channel utilization factor, and average delay. These measures are not usually appropriate for characterizing the properties of real-time channel access protocols. In a real-time access protocols it is important that messages arrive at their destination(s) within some specified periods of time after they have been sent to access the channel. Unfortunately, the existing measures for evaluation of access policies have little to characterize this property. For example, a real-time access protocol which fails to transmit in time a large number of messages with short deadlines and high traffic, could indeed perform poorly despite having a good (low) average delay characteristic or a high channel utilization factor, or even a relatively low bounded average delay.

This lack of appropriate measures for evaluating real-time access protocols is basically due to a more general lack of concern for real-time applications in most existing access protocols. However, we have found an increasing concern in the literature in this regard. In fact, some relevant evaluation measures have been proposed. Franta and Bilodeau | FRA80 | have used average delay for each communication interface (CIU) which is actually a node of network, for analyzing a prioritized CSMA protocol based on staggered delays. The assumption made has been that the CIU's have different priorities. A more relevant measure has been introduced for evaluation of message-based priority protocol in | CHL80 |, | TOB80 |. These papers consider CSMA protocols with message-based priorities for environments where there are a finite number of priority classes of messages. The measure used for evaluation is average delay for each prioriy class of messages.

However these priority access protocols are not suitable for most real- time environments and neither are the measures utilized for their evaluation. The definition of some appropriate measures such as the ones proposed in Section 4.3 is another part of our research in this section.

4.4.3. System-wide deadline oriented scheduling with unpredictable waiting times

In order to discuss the ability of existing scheduling algorithms to meet our promptness requirements, it is necessary to express the message transmission scheduling problem in terms currently used in scheduling theory. Our hypotheses are the following ones :

1) When in the free state a SMAP process gets hold of message, if any, ahead of its SQ(S) queue at current time t, t is message m **earliest transmission start time.**

2) Message m **latest transmission start time** is its deadline T(m). T(m) is imposed to SMAP and should SMAP fail to respect it, message m is rejected.

3) Message m transmission time is c(m). c(m) is a fixed value given message m length and the channel capacity. We may legitimately include DMAP service time in c(m) as it was done in Section 3.1, since DMAP service is performed immediatly upon m reception. So as to remain consistent with scheduling terminology, let us call c(m) the **transmission service time.**

4) Assuming the channel access protocol is of the CSMA-DCR (Deterministic Collision Resolution) type, we are authorized to state the following : Any collision resolution is terminated within an upperbounded period of time DCR(max) that depends of the collision resolution protocol overhead, the maximum packet length and the maximum possible number of sources in the system.

As a first consequence of this, given the time T at which SMAP attempts message m transmission, the channel may be busy in collision resolution state. From the CSMA-DCR epoch concept, it results that SMAP has to wait the epoch termination, that is a **maximum** period of time equal to DCR(max). In reality, since SMAP is in the free state, it has no message implied in the epoch and the maximum waiting time is only | DCR(max)-L/D | if L is the authorized message maximum length in the system and D the channel capacity. We will neglect L/D in the following, assuming the station number is in the order of magnitude of 50 which sounds reasonnable.

A second consequence is that , when channel is idle and SMAP attempts message m transmission, m may collide with other messages and enter an epoch. SMAP has no possible knowledge of the message transmission ordering within the epoch since it does not know the collided message relative priorities whatever these latter happen to be for the collision resolution mecanism. We can only state that, within this epoch, m will not start to be transmitted before a period of time DCR(m) with

$$DCR(m) \leqslant DCR(max) - c(m).$$

The result of these two observations is that the sum of these two waiting times for message m - we call it message m **channel access delay** - is Da, and the following relation holds :

$$0 \leqslant Da \leqslant 2 * DCR(max) - c(m) \qquad \text{Rel 1}$$

Da is equal to zero when, at the time T at which SMAP first attempts to transmit m, the channel is free **and** message m does not collide.

In conclusion to this analysis and in terms of scheduling, we are faced with the following hypothesis : The environment is that of a **one machine shop,** where the channel is the machine and the tasks are the message transmissions. A task has an unpredictable **earliest start time equal to its arrival time** : Message m earliest start time is t(m). Tasks have an unpredictable but upperbounded **execution time** : a message transmission time is c(m), it depends of the message length. The maximum execution time corresponds to messages maximum length. Tasks **latest start times** distribution law is unpredictable : A message transmission latest start time T(m) depends of its arrival time and the E(m) value assigned to the SendMessage (m) primitive (Section 3.1). Finally, due to the geographical distribution of the sources, there is an **unpredictable,** but upperbounded **waiting time** (2 * DCR(max)) between the time at which a task is scheduled and the time at which it is processed : that is between a transmission attempt and its start (the **channel access delay**).

4.4.4. Existing work in the field

| CON67 | and | MOK78 | have established that there exists no scheduling algorithms that guarantee no lateness when either start times, execution times or latest start times are unpredictable.

Very few algorithms report consider waiting times : they generally calculate a task start time with either the knowledge that the machine is idle at that time or that preemption of the executing task is allowed. This is because schedulers never compete with one another as in our case. Nevertheless some attempts has been made in the context of job shop scheduling where the machines set up times cannot be accuratly determined by schedulers. This set up time is exactly equivalent to our channel access delay.

| ZHA84 | introduce a weighted resources utilization ratio in the task choice function. But the resource utilization ratio to depends solely on local tasks requests. In our case, the channel load depends also on distributed sources pending and future requests.

| EIL68 | majorates tasks specified deadlines when the machine utilization ratio increases. This would correspond to a "dynamic" T(m) update which would mean that the SendMessage primitive user has no a priori insurance of the primitive execution time. This cannot cope with promptness constraints.

| GER66 | modifies the task execution time according to the machine load by using machine backloggs and observes through simulation that this may provide a lower performance that a simple slack-time scheduling rule where tasks are scheduled in the non decreasing order of their latest start time. The explanation for this is that time discrepancy between the execution time tuning and the machine load variations make optimal scheduling unachievable.

An other aspect of | GER66 | and | ZHA84 | is of major interest for us : They observe through simulation that very simple heuristics such as limited reordering of currently scheduled tasks and machine **idle time** artificial introduction increases the performance of any scheduling rule such as slack time and due date rules in the order of 20 %.

| MIN84 | relates of a distributed real time transaction system where C.P.U. allocation to local requests is scheduled with maximum waiting times. The propose heuristic shows clearly that it tends to minimize request rejections due to promptness control.

| BER75 | shows that introducing waiting time in both deadlines and execution time updates minimizes the shop termination time and the job lateness, but this is not our performance criteria.

These results must be considered only as a possible field for further investigations since they consider conservative shops in which no promptness control exists, with the exception of | ZHA84 |.

4.4.5. System-wide transmission scheduling and promptness control

The system-wide scheduling algorithm that was introduced in Section 3.2.2 consists in enforcing a tunable waiting delay W between the time t at which message m enters SMAP and the time T at which SMAP first attempts to transmit m, which we note

$W = T - t.$

As stated in section 3.2.2, we expect scheduled Ws to minimize the number of rejected messages due to promptness control. Let us now investigate precisely the impact of W with regard to this objective.

Let us assume that SMAP correctness is measured by a promptness control at the time Ta at which a message effectively accesses the channel. From the above observations on CSMA-DCR behaviour, it comes that

$Ta = T + Da.$

Hence message m passes successfully this promptness control iff

$T + Da = \leqslant T(m).$ Rel 2

Since

$T = t + W,$ Rel 3

it follows that

$t + W + Da \leqslant T(m)$ Rel 4

is the condition under which m is not rejected.
Clearly, the maximum value for W is $| T(m)-t |$ and it is an acceptable one iff Da is equal to zero, which can be written

$W = K * | T(m)-t|$ With $0 \leqslant K \leqslant 1.$ Rel 5

Then, Rel 3 and Rel 5 are equivalent to the formula given in Section 3.2.2, namely

$T = t + K * | T(m)-t |.$

Let us now focus on the relation between K, the SendMessage primitive arrival times and requested execution times (i.e. respectively the t(m) and E(m) distributions) and the system performance.

For message m, the deadline calculation provided in Section 3.2.1, gives

$T(m) = t(m) + E(m) - c(m) - d.$ Rel 6

Therefore, satisfying Rel 4 can be written, using Rel 5 and Rel 6,

$t + K * | T(m)-t | + Da \leqslant t(m) + E(m) - c(m) - d$ Rel 7

which can also be written

$K * | T(m)-t | + Da \leqslant E(m) - | t-t(m) | - c(m) - d.$ Rel 8

This formulation enables to discuss throughly the impact of K on the promptness control because it expresses the relation between K and

- Da, the channel access delay for m,
- E(m), the specified execution time of the SendMessage (m) primitive,
- $|t - t(m)|$, the delay m spends either waiting in UQ(S) and SQ(S) or being served in the SLP process. The waiting time in SQ(S) depends of the channel activity up to time t.
- c(m), message m transmission service time,
- d, the DLP process service time.

Clearly Rel 8 is satisfied for any values of $0 \leqslant K \leqslant 1$ if

$E(m) > | t-t(m) | + c(m) + d + Da.$

Let us examine the possible, if any, maximum value E (max) for E(m).

c(m) and d are constants for m.
Unfortunatly, Da, as we have explained, may reach | 2 * DCR(max) - c(m)|. Moreover, | t-t(m),t | depends of the message arrival time and deadline distributions during the period of time |t(m)-t | since SLP and SMAP process messages in UQ(S) and SQ(S) respectively in the order of increasing deadlines.

The question is to determine whether there exists an E(m) which is both maximal and an acceptable value and avoids rejections. The answer is that it is impossible to determine a maximal value of E(m) because of the unpredictable message arrival time distribution. The proof can be established as follows : At message m arrival time t(m) and whatever this time value is, there may be p(t(m)) messages in SQ(S) with lower deadline values. Each of these p(t(m)) messages have to be transmitted before m and each of them may have to wait |2 * DCR(max)| before it effectively starts to be transmitted. Hence E(max) should include |2 * p(t(m)) * DCR(max)| ; Assuming that a maximal value of p(t(m)) can be fixed, it is obviously incompatible with promptness in real-time environments. Hence, we assume in the following that E(m) is allowed to be smaller than E(max).

Since SMAP does not master | t-t(m) |, an apprently secure approach is to assign to K a small value, that is near or equal to zero. This rises matters for discussions, as stated threafter :

Assuming **K is equal to zero** for all stations means that, regardless of the deadline distribution, all free SMAPs at time t that have at least one message pending in their SQ(S) do attempt message ahead of SQ(S) transmission, which tends to provoke one or two colisions. As a consequence of this, a message m with rank r within the collision resolution algorithm message ordering may be rejected, whereas messages with much larger deadline value than m's and smaller rank than r are transmitted. As said in Section 3.2.2, values of K should distribute the transmission first attempt times in the order of message increasing deadlines. We will come back to this argument after we have pointed out a second observation related to the **local** impact of small values (including zero) for K.

Suppose, for the sake of simple wording, that K is set to zero locally, whatever is its value at other sources. Then at time t, SMAP attempts to transmit message m ahead of its SQ(S), if any. Assume that while SMAP processes m, a new message m' is deposited in SQ(S). SMAP remains busy with m transmission for a periode of time B(m) whose lower and upper bounds are respectively c(m) and |2 * DCR(max)|, but whose effective value is not predictable. It may happen that m' deadline issues before | t+B(m) |, in which case m' is rejected.

Now, setting K at its maximum value, namely 1, would not change the scenario since, as stated in Section 3.2.2, we restrict ourselves to the case where SMAP is **not preemptible**. As a consequence, K=1 would **maximize** the waiting time for m' and therefore increase m' rejection probability. Thus, we advise to have K equal to zero or some relatively small value in a source that might transmit emergency messages (T(m) not significantly different from t(m)).

Conversely, when SMAP is preemptible, K equal to 1 or a relatively large value increases the probability of transmitting emergency messages in time. That is not to say it minimizes the rejection number, but the demonstration for this assertion is out of this report context.

Suppose K has **different values** in different sources and moreover that the values assigned to K increase with message deadlines. Suppose messages are effectively transmitted in the order of their increasing deadlines. Such a system-wide scheduling is equivalent to a pure deadline ordering which is proved to produce the minimum task lateness but not the minimum number of late tasks.

Fortunately, since in our case, promptness control spares channel loading with late messages, one can expect such a policy for K to provide better results in terms of rejected messages. Nevertheless it is well known that when the requested throughput exceeds the capacity, the SPT (Shortest Processing Time) rule provides the minimum number of late tasks | CON67 |. Unfortunately, this result relies on predetermined execution times excluding unpredictable waiting times. Due to messages impredictable deadlines and promptness control effect, we have no mathematical proof that, when the requested throughput exceeds the channel capacity, transmiting the shortest messages first could provide less rejections than a deadline dependant ordering. Should it be the case, we assume that we must respect deadlines prior to rejection number because it reflects the emergency notion. Fortunately, **emergency** messages are rather short in most cases. More generally, for a relatively big proportion in real-time traffic, message lengths and message expected transmission delays vary similarly.

The last discussion on K concerns the impact of **K being equal to 1** or a relatively large value. Clearly this tends to attempt message first transmissions at a time very near their deadline, which means that the rejection probability increases with channel access delay (which is not predictable).

The conclusions we can draw from these observations an K impact are the following requirements,

1) Avoid K = constant = 0 for all stations,
2) Let K = 0 if, during a period of time, on a given source, message arrival times and deadlines do not vary in correlation, (This is valid specially for non preemptible SMAPs).
3) Keep K small when local requested throughput increases,
4) Let K grow with dealines so as to tend to transmit messages in the order of their increasing deadlines,
5) Avoid K = 1 when collisions are frequent.

Clearly, K tuning depends on the traffic pattern (arrival time and deadline distributions as well as message lengths).

Nevertheless we propose to study the following calculation methods for K :

. K is dependant of the local SQ(S) state :

.. $K = \alpha * (T1/(t+E(max))$ where

- T1 is equal to message ahead of SQ(S) deadline,
- α is an arbitrary tuning parameter that can make K as small as possible.

Such a calculation meets requirements 1,4 and 5

.. $K = \alpha * (1/n)$ with

- n equal to the number of waiting messages in SQ(S).

Such calculation meets requirements 1 and 5. It makes K equal to 1 when there is only one message in SQ(S) whiche may acceptable for large values of . Also, if n messages in SQ(S) have very large deadline values, K needs not be too small. In consequence a better calculation might be :

.. $K = \alpha * (1/n) * (T1 / (t + E(max))$ or

.. $K = \alpha * (1/n) * (Tj / (t + E(max))$ with

- j a given maximum number defining the j^{th} message in SQ(S).

Such calculations meet requierements 1, 3, 4, and 5.
For given values for j and α and given the traffic pattern it may meet requirement 2.

K may be set to be a constant **randomly** chosen for the different stations. Although this does not specifically meet the above requirements, it may perform as well as the above calculations for specific source traffic pattern, typically for correlated bursts, in which different sources wish to send at the same time messages with same deadlines.
Evidently, in this case K must be made equal to zero for emergency messages.

K is a local variable whose actual value is tuned **according to the channel access delay variations** as they may have been observed in the relatively short past time. Again one of two following decisions may be taken : Either K is decreased when the channel access delay increases, implying that as the channel load increases, SMAP, attempt transmissions as early as possible to avoid local rejections. Since collisions may occur, this is not guaranteed. Or reversely, K is increased to leave a chance to transmit messages in their deadline order, should t (m) and T (m) distributions not be correlated. (This only holds when SMAP is preemptible).

4.4.5. Conclusion and further work

Quite evidently there is no scheduling policy that can guarantee that all messages can be transmitted before their deadline, given our hypothesis.

Nevertheless, given that the expected order of magnitude of collision detection resolution overhead can grow from 25 to 50 % of the effective channel capacity, it is expected from the global scheduling to distribute transmission attempts so as to reduce the number of collisions and thus increase the channel effective throughput as defined in Section 4.1.2.

Our goal in future work is to observe performance results for different traffic patterns through simulations. We expect them to allow us to observe the effective SendMessage primitive execution time as well as the rejection distributions with regards to messages arrival time, deadline, static priority, and length distributions. Using different calculations of K for a given traffic should allow to appreciate K efficacity in minimizing the rejection number.

5. CONCLUSION

The work described in this report constitutes a first step in the study of problems and solutions related to real-time local networks and it is hoped that this report will provide a foundation for initiating the major efforts needed to resolve important design and modeling issues in this area.

REFERENCES

| BAR64 | P. Baran, "On distributed communications", Rand Corporation Series Reports, Santa Monica, August 1964, 453 p.

| BER75 | W.L. Berry and V. Rao, "Critical ratio scheduling : an experimental analysis", Management Science, Vol. 22, n° 2, Oct. 75, pp. 192-201.

| BEY81 | B. Beyaert, G. Florin, P. Lonc and S. Natkin, "Evaluation of computer dependability using stochastic Petri nets", in Proc. 1981 11th Int. Symp. on Fault-Tolerant Computing, Portland, ME, June 1981, pp. 66-71.

| CAP79 | J.I. Capetanakis, "Generalized TDMA : the multi-accessing tree protocol", IEEE Trans. on Communications, Com-27, October 1979, 1476-1484.

| CHL79 | I. Chlamtac, W.R. Franta, K.D. Levin, "BRAM : the broadcast recognizing access method", IEEE Trans. on Communications, Com-27, n°8, August 1979, 1183-1190.

| CHL80 | I. Chlamtac, W.R. Franta, "Message based priority access to local networks", Computer Communications, Vol. 3, 2, April 1980, 77-84.

| CON67 | R.W. Conway, W.L. Maxwell, L.W. Miller,"Theory of scheduling", Addison-Wesley publishing Co, 1967.

| DER67 | D. Derville et al., "Le système Haliotis", RIRO Journal n°6, 1967, 3-25 (in French).

| EIL68 | S. Eilon and D.J. Cotteril, "A modified SI rule in job shop scheduling", The International Journal of Production Research, 1968, Vol. 7, n° 2, pp. 135-145.

| FAR73 | D.J. Farber et al. "The distributed computing system", Proceedings 7th Annual IEEE Int. Conference, Feb. 1973, 31-34.

| FAY83 | G. Fayolle et al., "The evaluation of packet transmission characteristics in a multi-access resolution protocol", INRIA Research Report n° 245, October 1983, 19 p.

| FRA80 | W.R. Franta, M. Bilodeau, "Analysis of a prioritized CSMA protocol based on staggered delays", Acta Informatica, Vol. 13, Fasc. 4, 1980, 299-324.

| GER66 | W.S. Gere, "Heuristics in job shop scheduling", Management Science, vol.13, n°3, Nov. 1966, pp. 167-190

| GOL83 | Y.I. Gold, W.R. Franta, "An efficient collision-free protocol for prioritized access control of cable or radio-channels", Computer Networks 7, 1983, (North-Holland), 83-98.

| IID80 | I. Iida, M. Ishizuka, Y. Yasuda and M. Onoe, "Random access packet switched local computer network with priority function", in Proc. Nat. Telecommun. Conference, Houston, Texas, Dec. 1980, pp. 37.4.1-37.4.6.

| IEE82 | International Electrical and Electronical Engineers, Project 802, "Local Network Standards, Draft C", May 17, 1982, 698 p.

| ISO80 | International Standard Organization, "Reference Model of Open Systems Interconnection", Document ISO/TC97/SC16/N227

| KLE75 | L. Kleinrock, F.A. Tobagi, "Packet switching in radio channels : Part I - Carrier sense multiple-access modes and their throughput-delay characteristics", IEEE Trans. Commun., Vol. COM-23, December 1975, 1400-1416.

| KLE76 | L. Kleinrock, "Queueing systems", John Wiley & sons eds., 1976

| KUR 83 | J.F. Kurose et al., "Controlling window protocols for time-constrained communication in a multiple access environment", ACM Sigcomm, Vol. 13, 4, October 1983, 75-84.

| LAM82 | L. Lamport, P.M. Melliar-Smith, "Synchronizing clocks in the presence of faults", SRI International, March 1982, 26 p.

| LEL77 | G. Le Lann, "Distributed systems-Towards a formal approach", Proceedings IFIP Congress, (North-Holland), Aug. 1977, 155-160.

| LEL83a | G. Le Lann, "On real-time distributed computing", invited paper, Proceedings IFIP Congress, (North Holland), Sept. 1983, 741-753.

| LEL83b | G. Le Lann, "Deterministic multiple access protocols for real-time local area networks", INRIA Research Report n° 246, October 1983, 13 p.

| LEM78 | B. Lemaire, "Contribution à l'étude des systèmes avec attente - Application à des problèmes d'informatique", thèse d'Etat ParisVI, 247 pages.

| MAS81 | J.L. Massey, "Collision-resolution algorithms and random-access communications", in Multi-User Communication Systems (Ed. G. Longo), Springer-Verlag CISM n° 265, 1981, 73-137.

| MET76 | R.M. Metcalfe, D.R. Boggs, "Ethernet : distributed packet-switching for local computer networks", Communications of ACM, vol. 19, 7, July 1976, 395-404.

| MEY78 | J.F. Meyer, "On evaluating the performability of degradable computing systems", Proc. 1978 Int. Symp. on Fault-Tolerant Computing, Toulouse, France, June 1978, 44-49.

| MEY80 | J.F. Meyer, "On evaluating the performability of degradable computing systems", IEEE Trans. on Computers, vol. C-22, Aug. 1980, 720-731.

| MEY82 | J.F. Meyer, "Closed-form solutions of performability", IEEE Trans. on Computers, C-31, July 1982, 648-657.

| MEY85 | J.F. Meyer, A. Movaghar, W.H. Sanders, "Stochastic activity netwarks : Structure, behavior, and application, "in Proc. International Workshop or timed Petri Nets, Torino, Italy, July 1985.

| MIN84 | P. Minet and S. Sedillot, "lutegration of real-time and consistency contraints in distributed databases", INRIA. SCORE internal report, BAS.I.004. Sept. 84.

| MOK78 | A. Kan Mok, M.L. Dertouzos, "Multiprocessor scheduling in hard real-time environment", Proc. of the 7th Texas Conference on Computing Systems, Nov. 78, pp. 5.1-5.12.

| MOL81 | M.L. Molle, "Unifications and extensions of the multiple access communications problem", UCLA report n° CSD-810730 (Ph.D. thesis), July 1981, 139 p.

| MOV84 | A. Movaghar, J.F. Meyer, "Performability modeling with stochastic activity networks," in Proc. 1984 Real-Time System Symp., Austin, Tx, Dec. 1984.

| MOV85 | A. Movaghar, "Performability modeling with stochastic activity networks, " PhD Thesis, University of Michigan, Ann Arbor, MI, 1985.

| ONO78 | M. Onoe, Y. Yasuda and M. Ishizuka, "A random access packet communication system with priority function - priority Ethernet", in Proc. Nat. Conv. Inform. processing Soc., Japan, August 1978, paper 3A-1.

| POW81a | D.R. Powell, "Réseaux locaux de commande-contrôle sûrs de fonctionnement", Thèse d'Etat, INPT, October 1981, 212 p.

| POW81b | D.R. Powell, "NRThormance evaluation and comparison of dependable channel access techniques for locally-distributed computing systems", Proc. 2nd. ICDCS, April 1981, 256-270.

| ROM81 | R. Rom, F.A Tobagi, "Message-based priority functions in local multi-access communication systems", Computer Networks, 1981, 273-286.

| SCI83 | SCICON-INRIA, "Robust Real-Time Distributed Systems", Feasibility study, final report to the Commission of the European Communities, Brussels, Feb. 1983, 152 p.

| TOB80 | F.A. Tobagi, R. Rom, "Efficient round-robin and priority schemes for unidirectional broadcast systems", in Local Networks for Computer Communications (Eds. A. West, P. Janson), North-Holland/IFIP, 1980, 125-138.

| TOB82 | F.A. Tobagi, "Carrier-sense multiple access with message-based priority functions", IEEE Trans. on Communications, Com-30, January 1982, 185-200.

| TOW82 | D. Towsley, G. Venkatesh, "Window random access protocols for local area networks", IEEE Trans. on Computers, C-31, 8, August 1982, 715-722.

| VAL83 | J.C. Valadier, D.R. Powell, "On CSMA protocols allowing bounded channel access times", LAAS report n° 83.054, September 1983, 18 p.

| ZHA84 | W. Zhao, K. Ramamritham and J. Stankovic, "Scheduling tasks with resource requirements in hard real-time systems", Internal report, May 1984, University of Massachussetts, Amherst, USA.

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique